使用App Inventor 2开发UI界面丰富、功能强大的应用

- 介绍App Inventor 2的源码
- 介绍Android的相关知识
- 学会App Inventor 2自带组件的功能扩展
- 掌握App Inventor 2插件的开发
- 提供丰富的开发样例

App Inventor 2
定制与二次开发

强 增 ◎ 编著





机械工业出版社
CHINA MACHINE PRESS



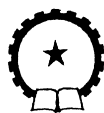
VISIT…

LANZAROTE

Caliente.com

App Inventor 2 定制与二次开发

强　增　编著



机 械 工 业 出 版 社

本书介绍了App Inventor 2开发涉及的Android相关知识，App Inventor 2的源码、自带组件的功能扩展和插件的开发，并提供了大量实例，使读者对App Inventor 2开发相关知识有一个全面了解，能够对App Inventor 2进行定制与二次开发，并在此基础上，能够以更专业和高效的方式开发出UI界面更丰富、功能更强大的应用。

本书的目标读者是从事App Inventor 2开发教育、培训工作的教师和培训人员，学习App Inventor 2开发的学生和开发人员等。无论是App Inventor 2的初学者，还是中高级读者，本书都是一本不可多得的开发秘籍。

图书在版编目（CIP）数据

App Inventor2 定制与二次开发 / 强增编著．—北京：机械工业出版社，2018.6

ISBN 978-7-111-59828-2

Ⅰ．①A… Ⅱ．①强… Ⅲ．①移动终端－应用程序－程序设计
Ⅳ．①TN929.53

中国版本图书馆CIP数据核字（2018）第088371号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
策划编辑：汤 枫 责任编辑：汤 枫
责任校对：张艳霞 责任印制：常天培

北京圣夫亚美印刷有限公司印刷

2018年6月第1版•第1次印刷
184mm×260mm•15.75印张•376千字
0001－2500册
标准书号：ISBN 978-7-111-59828-2
定价：59.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
服务咨询热线：（010）88379833
读者购书热线：（010）88379649
封面无防伪标均为盗版

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
教育服务网：www.cmpedu.com
金 书 网：www.golden-book.com

前　言

2017 年 7 月 20 日国务院印发了《新一代人工智能发展规划》，其中明确指出人工智能成为国际竞争的新焦点，应逐步开展全民智能教育项目，在中小学阶段设置人工智能相关课程，逐步推广编程教育，建设人工智能学科，培养复合型人才，形成我国人工智能人才高地；并强调大力加强人工智能劳动力培训，广泛开展人工智能科普活动，鼓励社会力量参与寓教于乐的编程教学软件、游戏的开发和推广。

此次规划的出台，必将加快编程教育进入中小学和职业教育课堂的实施进度，从而促进相关行业的发展。

App Inventor 2 是起源于 Google，后由 MIT（麻省理工学院）开发的一款图形化编程工具，它抛弃了复杂的程序代码，只需要拖动和排列积木形状的模块就可以编写各种应用，非常适合学生和非专业软件开发人员使用。

Google 也在和各教育部门合作，大力推广 App Inventor 2 编程开发，已经有许多学校、大学和社会培训机构开设了 App Inventor 2 编程课程。

目前介绍 App Inventor 2 开发相关的书籍很多，但基本是单纯介绍使用 App Inventor 2 开发应用；App Inventor 2 自身虽然提供了许多组件供开发应用使用，但这些组件或多或少存在某些缺陷或功能缺失，使用不是很方便，在开发应用的时候，无论是应用的 UI 界面，还是功能，都很受限制；而且用 App Inventor 2 开发应用还需要了解一些 Android 相关知识，这些书籍也都没有涉及。

有感于此，编者编写的这本书介绍了 App Inventor 2 开发涉及的 Android 相关知识，App Inventor 2 的源码、自带组件的功能扩展和插件的开发，并提供了大量实例，使读者对 App Inventor 2 开发相关知识有一个全面了解，能够对 App Inventor 2 进行定制与二次开发，并在此基础上，能够以更专业和高效的方式开发出 UI 界面更丰富、功能更强大的应用。

本书的章节分为以下几部分。

第一部分（第 1 章）：App Inventor 2 源码的获取和编译。

第二部分（第 2～5 章）：Android 知识介绍（Activity、Intent、Broadcast 和权限）。

第三部分（第 6 章）：App Inventor 2 中使用的注解介绍。

第四部分（第 7～18 章）：App Inventor 2 自身组件的定制（Screen 的定制、图像组件的定制、标签的定制、列表显示框的定制、对话框的定制、输入框的共性定制、密码输入框的定制、文本输入框的定制、Web 浏览框的定制、音频播放器的定制、视频播放器的定制、画布的定制、文件管理器的定制、微数据库组件的定制和 Web 客户端的定制）。

第五部分（第 19 章）：插件的开发。

第六部分（第 20 章）：综合实例开发。

由于编者水平有限，编写过程中难免存在不足和缺陷，欢迎广大读者和专家学者批评指正。

编　者

目　录

前言
第 1 章　App Inventor 2 源码的获取和编译 …… 1
1.1　源码的获取 …… 1
1.2　源码的编译 …… 1
第 2 章　Activity 介绍 …… 6
2.1　创建 Activity …… 6
2.2　用户界面的实现 …… 7
2.3　启动 Activity …… 7
2.3.1　启动 Activity 不获得结果 …… 7
2.3.2　启动 Activity 获得结果 …… 8
2.4　Activity 生命周期 …… 10
2.4.1　生命周期的状态 …… 10
2.4.2　生命周期的回调函数 …… 10
第 3 章　Intent 介绍 …… 13
3.1　Intent 简介 …… 13
3.2　Intent 类型 …… 13
3.3　构建 Intent …… 14
3.4　显式 Intent 示例 …… 17
3.5　隐式 Intent 示例 …… 17
3.6　Intent 过滤器 …… 19
第 4 章　Broadcast 介绍 …… 20
4.1　Broadcast 机制概述 …… 20
4.2　BroadcastReceiver …… 20
4.2.1　静态注册 …… 20
4.2.2　动态注册 …… 21
4.3　系统广播 …… 21
第 5 章　权限介绍 …… 23
5.1　权限简介 …… 23
5.2　常用权限说明 …… 23
第 6 章　注解介绍 …… 25
6.1　DesignerComponent …… 25

6.2 SimpleObject …… 27
6.3 UsesPermissions …… 27
6.4 DesignerProperty …… 28
6.5 SimpleProperty …… 29
6.6 SimpleFunction …… 30
6.7 SimpleEvent …… 30
6.8 UsesBroadcastReceivers …… 31
6.9 ReceiverElement …… 32
6.10 IntentFilterElement …… 33
6.11 ActionElement …… 33
6.12 UsesLibraries …… 34
第 7 章 Screen 的定制 …… 35
7.1 功能菜单的修改 …… 35
7.1.1 原有菜单的修改 …… 35
7.1.2 增加设置新增菜单的属性 …… 37
7.1.3 新增菜单的实现 …… 38
7.2 增加暂停和恢复状态事件响应函数 …… 41
第 8 章 图像组件（Image）的定制 …… 44
8.1 增加支持的动画形式 …… 44
8.2 增加设置动画播放次数和时间的功能 …… 47
8.3 增加单击事件响应函数 …… 51
第 9 章 标签（Label）的定制 …… 53
9.1 增加允许单行或多行显示的属性 …… 53
9.2 增加允许被选中复制的属性 …… 54
9.3 增加单击事件响应函数 …… 54
9.4 增加跑马灯效果函数 …… 55
9.5 增加设置标签图片的函数 …… 56
9.6 增加富文本处理函数 …… 58
9.7 实现结果 …… 63
9.8 使用示例 …… 64
第 10 章 列表显示框（Listview）的定制 …… 73
10.1 修改显示字符串 …… 73
10.2 增加显示框的显示样式 …… 73
第 11 章 对话框（Notifier）的定制 …… 77
11.1 修改对话框的外观 …… 77
11.1.1 增加设置图标的属性 …… 77
11.1.2 增加修改对话框显示风格的属性 …… 79

11.2 增加显示函数…………………………………………………………………………82
11.3 修改日志功能…………………………………………………………………………87
11.3.1 日志功能介绍…………………………………………………………………87
11.3.2 增加设置日志开关属性………………………………………………………89
第 12 章 输入框的定制…………………………………………………………………92
12.1 输入框的共性定制……………………………………………………………………92
12.1.1 增加设置字符串长度的属性和核查函数……………………………………92
12.1.2 增加清空输入框的函数………………………………………………………94
12.2 密码输入框（PasswordTextBox）的定制……………………………………………95
12.3 文本输入框（TextBox）的定制………………………………………………………97
第 13 章 Web 浏览框（WebView）的定制……………………………………………99
13.1 增加拨号功能…………………………………………………………………………99
13.2 返回键功能的处理…………………………………………………………………101
13.3 增加加载网页事件响应函数………………………………………………………102
第 14 章 播放器的定制…………………………………………………………………105
14.1 音频播放器（Player）的定制………………………………………………………105
14.1.1 增加权限………………………………………………………………………105
14.1.2 增加获取数据函数……………………………………………………………105
14.1.3 增加播放函数…………………………………………………………………106
14.2 视频播放器（VideoPlayer）的定制…………………………………………………107
第 15 章 画布（Canvas）的定制………………………………………………………108
15.1 增加画图函数………………………………………………………………………108
15.2 增加绘制路径函数…………………………………………………………………109
15.3 增加滑动事件响应函数……………………………………………………………111
15.4 设置函数参数的默认值……………………………………………………………112
15.5 增加声明和中文字符串……………………………………………………………113
15.6 路径函数使用示例…………………………………………………………………117
第 16 章 文件管理器（File）的定制……………………………………………………118
16.1 增加文件处理函数…………………………………………………………………118
16.2 增加获取存储区信息的函数………………………………………………………120
第 17 章 微数据库组件（TinyDB）的定制……………………………………………121
17.1 存储功能介绍………………………………………………………………………121
17.2 增加设置数据文件名称和数据加密功能…………………………………………122
17.3 使用示例……………………………………………………………………………127
第 18 章 Web 客户端的定制……………………………………………………………130
18.1 增加 PATCH 请求函数………………………………………………………………130
18.2 增加设置 JSON 格式数据的属性……………………………………………………131

第 19 章　插件（Extension）开发 ······ 134
19.1　加密功能插件 ······ 134
19.1.1　插件的实现 ······ 134
19.1.2　插件的编译 ······ 136
19.1.3　插件的导入 ······ 136
19.1.4　插件的使用 ······ 137
19.2　对话框插件 ······ 137
19.2.1　插件的实现 ······ 137
19.2.2　插件的导入 ······ 138
19.3　获取设备和系统信息插件 ······ 139
19.3.1　插件的实现 ······ 139
19.3.2　插件的使用 ······ 142
19.4　监测设备电量状态插件 ······ 143
19.4.1　插件的实现 ······ 143
19.4.2　插件的使用 ······ 146
19.5　获取和监测设备网络状态插件 ······ 147
19.5.1　插件的实现 ······ 147
19.5.2　插件的使用 ······ 151
19.6　获取和监测设备飞行模式状态插件 ······ 152
19.6.1　插件的实现 ······ 153
19.6.2　插件的使用 ······ 155
19.7　系统设置插件 ······ 155
19.7.1　插件的实现 ······ 155
19.7.2　插件的使用 ······ 157
19.8　列表插件 ······ 158
19.8.1　列表创建 ······ 158
19.8.2　列表排序 ······ 159
19.8.3　列表查找 ······ 160
19.8.4　插件的实现 ······ 161
19.8.5　插件的使用 ······ 164
19.9　代数计算插件 ······ 165
19.9.1　矩阵的运算 ······ 165
19.9.2　复数的运算 ······ 166
19.9.3　插件的实现 ······ 167
19.9.4　插件的使用 ······ 171
19.10　几何计算插件 ······ 172
19.10.1　计算图形的面积 ······ 172

19.10.2 计算图形的体积 ······ 172
19.10.3 插件的实现 ······ 172
19.11 手电筒功能插件 ······ 175
19.11.1 插件的实现 ······ 175
19.11.2 插件的使用 ······ 177
19.12 手机号码校验插件 ······ 178
19.12.1 插件的实现 ······ 178
19.12.2 插件的使用 ······ 179
19.13 判断应用是否第一次运行插件 ······ 179
19.13.1 插件的实现 ······ 180
19.13.2 插件的使用 ······ 182
19.14 获取定位功能状态插件 ······ 182
19.14.1 插件的实现 ······ 182
19.14.2 插件的使用 ······ 184
19.15 创建 JSON 字符串插件 ······ 184
19.15.1 插件的实现 ······ 184
19.15.2 插件的使用 ······ 187
第 20 章 综合实例开发 ······ 189
20.1 抓青蛙游戏 ······ 189
20.1.1 游戏简介 ······ 189
20.1.2 游戏使用的素材 ······ 189
20.1.3 游戏使用的插件 ······ 189
20.1.4 游戏实现 ······ 190
20.2 画图应用 ······ 203
20.2.1 应用简介 ······ 203
20.2.2 应用使用的素材 ······ 204
20.2.3 应用使用的插件 ······ 204
20.2.4 应用实现 ······ 204
20.3 定位应用 ······ 225
20.3.1 应用简介 ······ 225
20.3.2 应用使用的素材 ······ 226
20.3.3 应用使用的插件 ······ 226
20.3.4 应用实现 ······ 227

第 1 章　App Inventor 2 源码的获取和编译

1.1　源码的获取

从 GitHub 网站上可以获取到 App Inventor 2 的源码，网址如下：https://github.com/mit-cml/ appinventor-sources。

在计算机上安装 git 后，使用以下命令从 GitHub 服务器上复制代码：

```
git clone https://github.com/mit-cml/appinventor-sources.git
```

App Inventor 2 源码还使用了 git submodule 机制，执行完 git clone 命令后，进入到源码的根目录/appinventor，然后执行：

```
git submodule update --init
```

这样才能获得全部代码。

1.2　源码的编译

App Inventor 2 的编译工具是 ant，计算机上需要有 ant 工具，并配置 ant 的环境变量，在 MAC 电脑上显示如下：

```
export ANT_HOME=/Users/ruwang/Downloads/apache-ant-1.10.1
export PATH=${PATH}:${ANT_HOME}/bin
```

在 App Inventor 2 的 GitHub 网页上，有如下说明：

```
You will need a full Java JDK (6 or 7, preferably from Oracle; JRE is not enough) and Python to compile and run the servers.
```

需要在计算机上安装 JDK6 或 JDK7，才能正常编译。

但 ant 的版本不同，需要的 JDK 也不同，如 1.10.1 版本就需要 JDK8，否则编译的时候，会报如下错误：

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/apache/tools/ant/launch/Launcher: Unsupported major.minor version 52.0
```

编译的时候，在 App Inventor 2 源码的根目录/appinventor 下，直接输入 ant 命令，会编

译所有源码，花费时间较长，实际往往不需要编译全部代码，只编译改动的模块就可以了。

在源码的根目录/appinventor下有个build.xml文件，在其中有许多target标签：

```
<target name="all">
    <ant inheritAll="false" useNativeBasedir="true" dir="appengine"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="blocklyeditor"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="common"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="buildserver"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="components"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="buildserver" target="PlayApp"/>
  </target>

   …
  <target name="MakeAuthKey">
    …

  <target name="comps">
  …
  <target name="extensions">
…
<target name="clean">
    <ant inheritAll="false" useNativeBasedir="true" dir="appengine" target="clean"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="blocklyeditor" target="clean"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="aimerger" target="clean"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="buildserver" target="clean"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="common" target="clean"/>
    <ant inheritAll="false" useNativeBasedir="true" dir="components" target="clean"/>
    <delete dir="build"/>
    <delete dir="reports"/>
  </target>
```

每个target标签中间说明了使用ant + target name时编译的模块，如输入如下命令：

ant all——编译所有模块，等同于只输入ant；

ant extensions——编译插件；

ant MakeAuthKey——编译AuthKey。

还有个特殊的编译命令：

ant clean——删除之前编译生成的build和reports文件夹。

需要先使用ant MakeAuthKey命令生成AuthKey后，才能使用其他命令编译。

使用ant命令编译后，还不能在本机运行App Inventor 2的开发环境，计算机上还需有appengine-java-sdk，同时也要配置环境变量：

```
export PATH=$PATH:/Users/ruwang/Downloads/appengine-java-sdk-1.9.54/appengine-java-sdk-1.9.54/bin/
```

在源码的根目录/appinventor下，运行如下命令：

```
/Users/ruwang/Downloads/appengine-java-sdk-1.9.54/bin/dev_appserver.sh  --port=8888 --address=  0.0.0.0
```

```
appengine/build/war/
```

当命令行窗口显示“信息：Dev App Server is now running”这行字符串时，表示此命令运行成功，如图 1-1 所示。

```
appinventor — java ◂ java -ea -cp ~/Downloads/appengine-java-sdk-1.9.54/lib/a...
eIo <clinit>
信息: Default GCS Bucket Configured from App Identity: app_default_bucket
十月 17, 2017 1:55:58 上午 com.google.appengine.api.datastore.dev.LocalDatastore
Service init
信息: Local Datastore initialized:
        Type: High Replication
        Storage: /Users/ruwang/Documents/code/learn/appinventor/appinventor-sour
ces/appinventor-sources/appinventor/appengine/build/war/WEB-INF/appengine-genera
ted/local_db.bin
十月 17, 2017 1:55:58 上午 com.google.appengine.api.datastore.dev.LocalDatastore
Service load
信息: Time to load datastore: 62 ms
十月 17, 2017 1:55:58 上午 com.google.apphosting.utils.jetty.JettyLogger info
信息: Started SelectChannelConnector@0.0.0.0:8888
十月 17, 2017 1:55:58 上午 com.google.appengine.tools.development.AbstractModule
 startup
信息: Module instance default is running at http://localhost:8888/
十月 17, 2017 1:55:58 上午 com.google.appengine.tools.development.AbstractModule
 startup
信息: The admin console is running at http://localhost:8888/_ah/admin
十月 17, 2017 9:55:58 上午 com.google.appengine.tools.development.DevAppServerIm
pl doStart
信息: Dev App Server is now running
```

图 1-1　dev_appserver.sh 命令运行结果

然后在源码的/appinventor/buildserver 目录下，运行如下命令：

```
        ant RunLocalBuildServer
```

当命令行窗口显示“[java] 信息：Server running”这行字符串时，表示此命令运行成功，如图 1-2 所示。

```
buildserver — java ◂ java -classpath ~/Downloads/apache-ant-1.10.1/lib/ant-laun...
Server main
     [java] 信息: App Inventor Build Server - Version: nls-1064-g3274ac6-dirty
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: App Inventor Build Server - Git Fingerprint: 3274ac6e091c275ce
921fea4673dfae6b73c1b44
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: Running at: http://10.0.34.123:9990/buildserver
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: Maximum simultanous builds = unlimited!
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: Visit: http://10.0.34.123:9990/buildserver/health for server h
ealth
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: Visit: http://10.0.34.123:9990/buildserver/vars for server val
ues
     [java] 十月 17, 2017 10:08:03 上午 com.google.appinventor.buildserver.Build
Server main
     [java] 信息: Server running
```

图 1-2　RunLocalBuildServer 命令运行结果

在浏览器的地址栏中，输入如下地址：http://localhost:8888，按回车键后，显示内容如图 1-3 所示。

Welcome to App Inventor!

Email

Password

Login

Set or Recover Password

Click Here to use your Google Account to login

中文 English





图 1-3 App Inventor 2 欢迎界面

系统语言默认是英文，可以单击中文按钮，将系统语言切换为中文。

单击图 1-4 所示登录文字标签，将显示图 1-5 所示的登录界面。

Click Here to use your Google Account to login

图 1-4 登录文字标签

Not logged in

Email: test@example.com

Sign in as Administrator

Log In Log Out

图 1-5 登录界面

单击“Log In”按钮，显示图 1-6 所示服务条款界面。

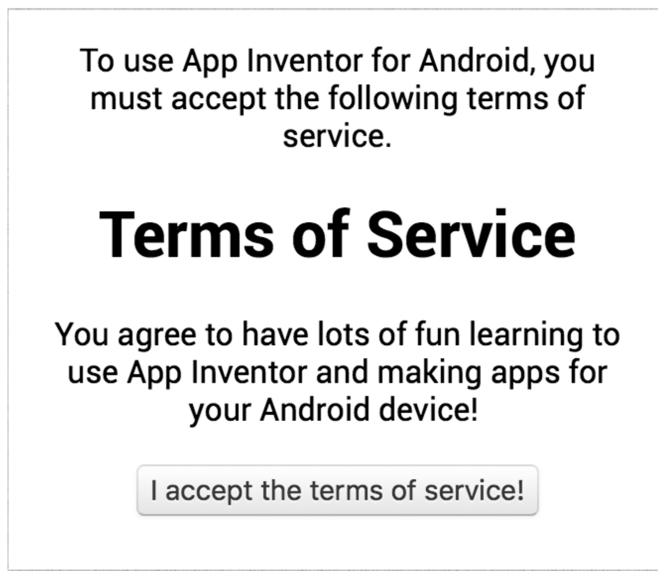


图 1-6　服务条款界面

单击“I accept the terms of service!”按钮就可以进入开发环境，开始 App Inventor 2 的编程之旅了。

App Inventor 2 是基于 Android 系统衍生的一个系统，无论是使用 App Inventor 2 开发应用，还是对 App Inventor 2 进行二次开发，都要用到许多 Android 开发的相关知识，下面介绍 Activity、Intent 和权限等相关知识。

第 2 章　Activity 介绍

Activity 是一个应用组件，用户可与其提供的屏幕进行交互，执行拨打电话、拍摄照片、发送电子邮件或查看地图等各种操作。

每个 Activity 都有一个用于绘制此 Activity 界面的窗口。窗口通常会充满屏幕，但也可小于屏幕并浮动在其他窗口之上。

一个应用通常由多个彼此松散联系的 Activity 组成。一般会指定应用中的某个 Activity 为“主”Activity，即首次启动应用时呈现给用户的那个 Activity。而且每个 Activity 均可启动另一个 Activity，以便执行不同的操作。

在 App Inventor 2 中，一个 Screen 就对应一个 Activity。Screen 的源码文件为 Form.java，其中有以下代码：

```
public class Form extends Activity
    implements Component, ComponentContainer, HandlesEventDispatching,
    OnGlobalLayoutListener {
…
}
```

Form 类是 Activity 的子类，所以 Screen 就对应着 Activity。

2.1　创建 Activity

创建 Activity 时，要实现 Activity 在其生命周期的各种状态之间转变（例如，创建 Activity、停止 Activity、恢复 Activity 或销毁 Activity）时系统调用的函数。

两个最重要的函数是：

```
①onCreate()
```

此函数必须实现。系统会在创建 Activity 时调用此函数。在实现过程中，应初始化 Activity 的必需组件。

```
②onPause()
```

系统将此函数作为用户离开 Activity 的第一个信号（但并不总是意味着 Activity 会被销毁）进行调用。通常在此函数内应确认在当前用户会话结束后仍然有效的任何更改（因为用户可能不会返回）。

2.2 用户界面的实现

Activity 的用户界面是由层级式视图（衍生自 View 类的对象）提供的。每个视图都控制 Activity 窗口内的特定矩形空间，可对用户交互做出响应。例如，视图可以是在用户触摸时启动某项操作的按钮。

可以利用 Android 提供的许多现成视图设计和组织布局。“小部件”是提供按钮、文本字段、复选框或仅仅是一幅图像等屏幕视觉（交互式）元素的视图。“布局”是衍生自 ViewGroup 类的视图，为其子视图提供唯一布局模型，如线性布局、网格布局或相对布局等。

还可以为 View 类和 ViewGroup 类创建子类（或使用其现有子类）来自行创建小部件和布局，然后将它们应用于 Activity 布局。

视图定义布局的最常见方法是借助保存在应用资源内的 XML 布局文件。这样，就可以将用户界面的设计与定义 Activity 行为的源代码分开维护。

可以通过 setContentView() 将布局设置为 Activity 的 UI，从而传递布局的资源 ID。也可以在 Activity 代码中创建新 View，并通过将新 View 插入 ViewGroup 来创建视图层次，然后通过将根 ViewGroup 传递到 setContentView() 来使用该布局。

2.3 启动 Activity

可以通过调用 startActivity()，并将其传递给描述想启动的 Activity 的 Intent 来启动另一个 Activity。

Intent 对象指定想启动的具体 Activity 或描述想执行的操作类型（系统会选择合适的 Activity，甚至是来自其他应用的 Activity）。Intent 对象还可能携带少量供所启动 Activity 使用的数据。

2.3.1 启动 Activity 不获得结果

在应用运行时，经常需要启动某个已知 Activity。可以通过使用类名创建一个显式定义想启动的 Activity 的 Intent 对象来实现此目的。

例如，可以通过以下代码让一个 Activity 启动另一个名为 LogInActivity 的 Activity：

```
Intent intent = new Intent(this, LogInActivity.class);
startActivity(intent);
```

应用可能还需要利用 Activity 数据执行某项操作，如打电话、打开网页或发短信等。在这种情况下，应用自身可能不具有执行此类操作所需的 Activity，因此可以改为利用设备上其他应用提供的 Activity 来执行这些操作。这便是 Intent 对象的真正价值所在。

开发人员可以创建一个 Intent 对象，对想执行的操作进行描述，系统会从其他应用启动相应的 Activity。如果有多个应用可以处理 Intent，则用户可以选择要使用哪一个。

在 App Inventor 2 的电话拨号器组件的源码文件 PhoneCallUtil.java 中，有如下代码：

```
public static void makePhoneCall(Context context, String phoneNumber) {
    if (null != phoneNumber && phoneNumber.length() > 0) {
        Uri phoneUri = Uri.parse("tel:" + phoneNumber);
        Intent intent = new Intent(Intent.ACTION_CALL, phoneUri);
        context.startActivity(intent);
    }
}
```

此函数用于调用设备上的拨打电话应用，当拨打电话应用响应此 Intent 时，会读取 Intent 函数第二个参数设置的电话号码拨打电话，并且当用户完成操作，退出拨打电话应用后，之前应用的 Activity 会恢复执行。

2.3.2 启动 Activity 获得结果

有时，可能需要从启动的 Activity 获得结果，也就是 A 启动 B，B 再返回数据给 A。在这种情况下，通过调用 startActivityForResult()（而非 startActivity()）来启动 Activity。要想在 A 中收到 B 返回的数据，需要实现 onActivityResult()函数。在 B Activity 中，使用 Intent 向 onActivityResult()函数返回结果。

在 App Inventor 2 的条码扫描器组件的源码文件 BarcodeScanner.java 中，有如下代码：

```
@SimpleFunction(description = "Begins a barcode scan, using the camera. When the scan " +
    "is complete, the AfterScan event will be raised.")
public void DoScan() {
  Intent intent = new Intent(SCAN_INTENT);
  if (!useExternalScanner && (SdkLevel.getLevel() >= SdkLevel.LEVEL_ECLAIR)) {
    // Should we attempt to use an internal scanner?
    String packageName = container.$form().getPackageName();
    intent.setComponent(new ComponentName(packageName, "com.google.zxing.client.android.
    AppInvCaptureActivity"));
  }
  if (requestCode == 0) {
    requestCode = form.registerForActivityResult(this);
  }
  try {
    container.$context().startActivityForResult(intent, requestCode);
  } catch (ActivityNotFoundException e) {
    e.printStackTrace();
    container.$form().dispatchErrorOccurredEvent(this, "BarcodeScanner",
      ErrorMessages.ERROR_NO_SCANNER_FOUND, "");
  }
}
```

```
@Override
public void resultReturned(int requestCode, int resultCode, Intent data) {
  if (requestCode == this.requestCode && resultCode == Activity.RESULT_OK) {
    if (data.hasExtra(SCANNER_RESULT_NAME)) {
      result = data.getStringExtra(SCANNER_RESULT_NAME);
    } else {
      result = "";
    }
    AfterScan(result);
  }
}
```

在 App Inventor 2 中的 Form.java 文件中有如下代码：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  Log.i(LOG_TAG, "Form " + formName + " got onActivityResult, requestCode = " +
      requestCode + ", resultCode = " + resultCode);
  if (requestCode == SWITCH_FORM_REQUEST_CODE) {
    // Assume this is a multiple screen application, and a secondary
    // screen has closed.  Process the result as a JSON-encoded string.
    // This can also happen if the user presses the back button, in which case
    // there's no data.
    String resultString;
    if (data != null && data.hasExtra(RESULT_NAME)) {
      resultString = data.getStringExtra(RESULT_NAME);
    } else {
      resultString = "";
    }
    Object decodedResult = decodeJSONStringForForm(resultString, "other screen closed");
    // nextFormName was set when this screen opened the secondary screen
    OtherScreenClosed(nextFormName, decodedResult);
  } else {
    // Another component (such as a ListPicker, ActivityStarter, etc) is expecting this result.
    ActivityResultListener component = activityResultMap.get(requestCode);
    if (component != null) {
      component.resultReturned(requestCode, resultCode, data);
    }
  }
}
```

使用扫描功能时，在 BarcodeScanner 类的 DoScan()函数中调用 startActivityForResult()函数启动扫描 Activity，扫描完成后，把得到的数据传递给 Form 类的 onActivityResult()函数，最终调用 BarcodeScanner 类的 resultReturned()函数处理扫描数据。（Screen 对应

Activity，条码扫描器组件附着在 Screen 上，所以把数据传递给 Form 类的 onActivityResult()函数）。

在 resultReturned()函数中，首先检查请求是否成功（如果成功，则 resultCode 将为 RESULT_OK），以及是否是指定请求返回的数据，也就是将 requestCode 与 startActivityForResult()函数发送的第二个参数匹配。只有请求成功，且 requestCode 匹配成功后，才处理 data 参数返回的数据。

可以通过调用 Activity 的 finish()函数来结束当前 Activity，还可以通过调用 finishActivity()函数结束当前之前启动的另一个 Activity。

在大多数情况下，不应使用这些方法显式结束 Activity。Android 系统会管理 Activity 的生命周期，因此无须开发人员编写代码结束 Activity。调用这些函数可能对预期的用户体验产生不良影响，因此只应在确实不想让用户返回此 Activity 实例时，才编写代码结束 Activity，而不依赖系统结束 Activity。

2.4 Activity 生命周期

2.4.1 生命周期的状态

Activity 基本上以三种状态存在。

1．继续

此种状态的 Activity 位于屏幕前台并具有用户焦点，可响应用户操作（也将此状态称作“运行中”）。

2．暂停

有一个部分透明或未覆盖整个屏幕的 Activity 显示在另一个 Activity 上方，位于屏幕前台并具有用户焦点，响应用户操作，此时位于下方的 Activity 仍可见，所处的状态就是暂停状态。暂停的 Activity 处于完全活动状态（Activity 对象保留在内存中，它保留了所有状态和成员信息，并与窗口管理器保持连接），但在内存极度不足的情况下，可能会被系统终止。

3．停止

此种状态的 Activity 被另一个 Activity 完全遮盖（该 Activity 目前位于“后台”）。已停止的 Activity 同样仍处于活动状态（Activity 对象保留在内存中，它保留了所有状态和成员信息，但未与窗口管理器连接）。不过，它对用户不再可见，在他处需要内存时可能会被系统终止。

如果 Activity 处于暂停或停止状态，系统会通过调用 finish()函数结束此 Acticity 或直接终止其进程，并将其从内存中删除。

将其结束或终止后，再次打开 Activity 时，必须调用 onCreate()函数重建。

2.4.2 生命周期的回调函数

Activity 生命周期如图 2-1 所示。

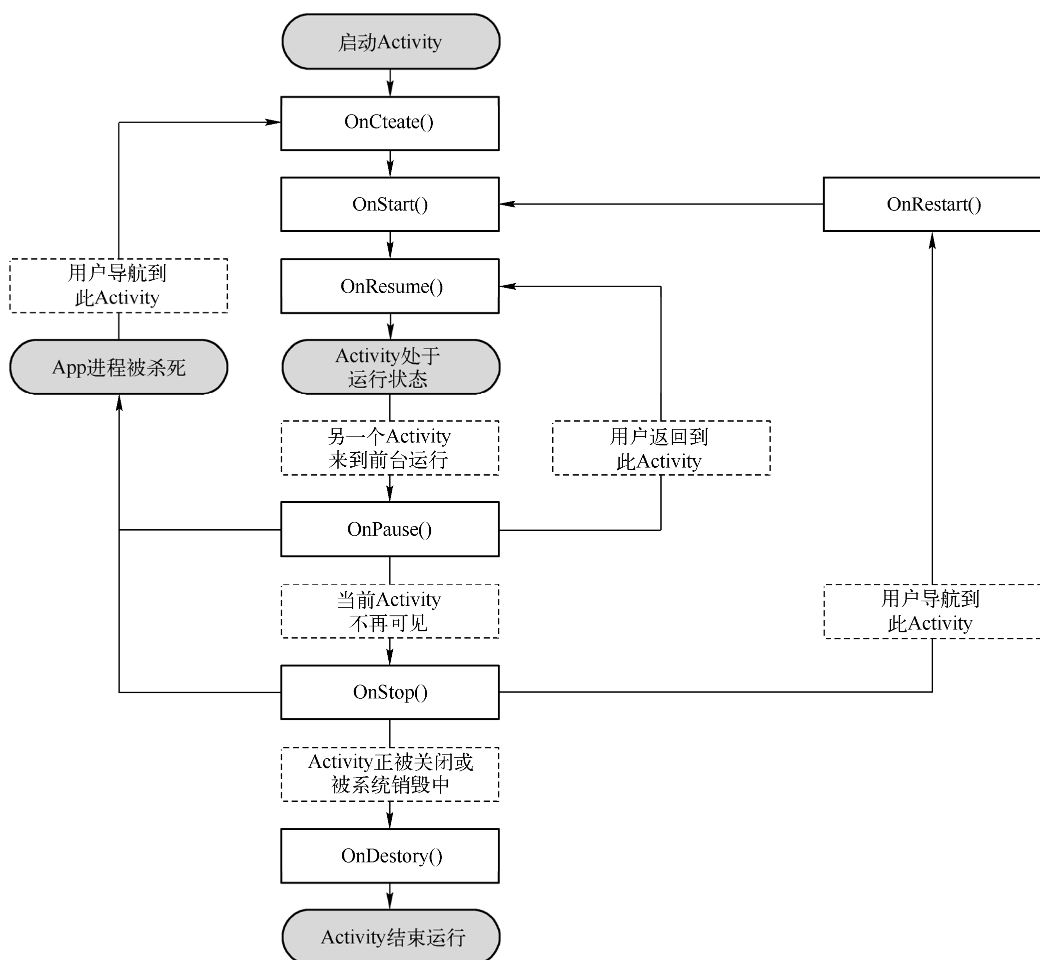


图 2-1　Activity 生命周期

表 2-1 列出了生命周期回调函数，其中对每一个回调函数做了更详细的描述，并说明了每一个函数在 Activity 整个生命周期内的位置，包括在回调函数完成后系统能否终止 Activity。

表 2-1　Activity 生命周期回调函数

函数	说明	是否能事后终止?	后接
onCreate()	创建 Activity 时调用。应该在此函数中执行所有正常的静态设置——创建视图、将数据绑定到列表等。系统向此函数传递一个 Bundle 对象，其中包含 Activity 的上一状态，不过前提是捕获了该状态。始终接 onStart()	否	onStart()
onRestart()	在 Activity 已停止并即将再次启动前调用。 始终接 onStart()	否	onStart()
onStart()	在 Activity 即将对用户可见之前调用。如果 Activity 转入前台，则后接 onResume()，如果 Activity 转入隐藏状态，则后接 onStop()	否	onResume() 或 onStop()

（续）

函数	说明	是否能事后终止?	后接
onResume()	在 Activity 即将开始与用户进行交互之前调用。此时，Activity 处于 Activity 堆栈的顶层，并具有用户输入焦点。 始终后接 onPause()	否	onPause()
onPause()	当系统即将开始继续另一个 Activity 时调用。此函数通常用于确认对持久性数据的未保存更改、停止动画以及其他可能消耗 CPU 的内容，诸如此类。它应该非常迅速地执行所需操作，因为它返回后，下一个 Activity 才能继续执行。如果 Activity 返回前台，则后接 onResume()，如果 Activity 转入对用户不可见状态，则后接 onStop()	是	onResume()或 onStop()
onStop()	在 Activity 对用户不再可见时调用。如果 Activity 被销毁，或另一个 Activity（一个现有 Activity 或新 Activity）继续执行并将其覆盖，就可能发生这种情况。 如果 Activity 恢复与用户的交互，则后接 onRestart()，如果 Activity 被销毁，则后接 onDestroy()	是	onRestart() 或 onDestroy()
onDestroy()	在 Activity 被销毁前调用。这是 Activity 收到的最后调用。当 Activity 结束（对 Activity 调用了 finish()），或系统为节省空间而暂时销毁该 Activity 实例时，可能会调用它。可以通过 isFinishing() 函数区分这两种情形	是	无

名为“是否能事后终止?”的列表示系统是否能在不执行其他 Activity 代码的情况下，在方法返回后终止承载 Activity 的进程。

有三个函数带有“是”标记：(onPause()、onStop() 和 onDestroy())。

由于 onPause() 是这三个函数中的第一个，因此 Activity 创建后，需要调用 onPause() 函数，然后才能终止进程；如果系统在紧急情况下必须恢复内存，则可能不会调用 onStop()和onDestroy()。因此，应该使用 onPause() 向存储设备写入至关重要的持久性数据（例如，用户编辑或输入的数据等）。不过，应对 onPause() 调用期间必须保留的信息有所选择，如果该方法的执行时间过长，会妨碍向下一个 Activity 的转变并且用户感觉到应用运行速度变慢。

在“是否能在事后终止?”列中标记为“否”的函数可从系统调用它们的一刻起防止承载 Activity 的进程被终止。

在 onPause() 执行完毕到 onResume() 被调用的这段时间内，系统可以终止 Activity。在onPause()被再次调用并返回前，将无法再次终止 Activity。

根据上述内容的描述，属于技术上无法“终止”的 Activity 仍可能被系统终止，但这种情况只有在无任何其他资源的极端情况下才会发生。

第 3 章　Intent 介绍

3.1　Intent 简介

Intent 是一个消息传递对象，可以使用它向其他应用组件请求操作。尽管 Intent 可以通过多种方式促进组件之间的通信，但其基本用例主要包括以下三种。

1．启动 Activity

通过将 Intent 传递给 startActivity()，可以启动新的 Activity 实例。Intent 描述了要启动的 Activity，并携带了任何必要的数据。

如果希望在 Activity 运行完成后收到结果，需要调用 startActivityForResult()。在 Activity 的 onActivityResult() 回调中，Activity 将结果作为单独的 Intent 对象接收。

2．启动服务

Service 是一个没有用户界面而在后台执行操作的组件。通过将 Intent 传递给 startService()，可以启动服务执行操作（例如，下载文件）。Intent 描述了要启动的服务，并携带了任何必要的数据。

3．传递广播

广播是任何应用均可接收的消息。系统将针对系统事件（例如，系统启动或设备开始充电时）传递各种广播。通过将 Intent 传递给 sendBroadcast()、sendOrderedBroadcast()或 sendStickyBroadcast()，可以将广播传递给其他应用。

3.2　Intent 类型

1．显式 Intent

显式 Intent 按名称（完全限定类名）指定要启动的组件。通常，开发人员会在自己开发的应用中使用显式 Intent 来启动组件，这是因为开发人员知道要启动的 Activity 或服务的类名。例如，启动新 Activity 以响应用户操作，或者启动服务以在后台下载文件。

创建显式 Intent 启动 Activity 或服务时，系统将立即启动 Intent 对象中指定的应用组件。

2．隐式 Intent

隐式 Intent 不指定特定的组件，而是声明要执行的常规操作，从而允许其他应用中的组件来处理它。例如，如需打开网页，则可以使用隐式 Intent，请求另一具有此功能的应用打开指定的网页。

图 3-1 描述了隐式 Intent 如何通过系统传递以启动其他 Activity。

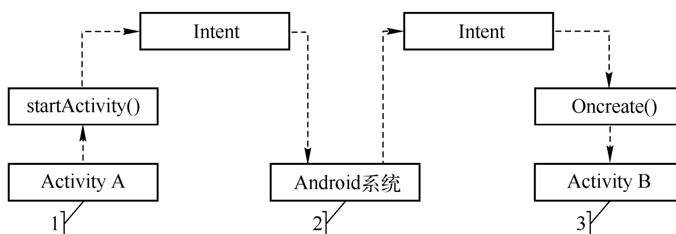


图 3-1 隐式 Intent 启动 Activity 示意图

1）Activity A 创建包含操作描述的 Intent，并将其传递给 startActivity()。

2）Android 系统搜索所有应用中与 Intent 匹配的 Intent 过滤器。

3）找到匹配项之后，系统通过调用匹配 Activity（Activity B）的 onCreate() 方法并将其传递给 Intent，以此启动匹配 Activity。

创建隐式 Intent 时，Android 系统通过将 Intent 的内容与在设备上其他应用的清单文件中声明的 Intent 过滤器进行比较，从而找到要启动的相应组件。如果 Intent 与 Intent 过滤器匹配，则系统将启动该组件，并向其传递 Intent 对象。如果多个 Intent 过滤器兼容，则系统会显示一个对话框，支持用户选取要使用的应用。

Intent 过滤器是应用清单文件中的一个表达式，它指定该组件要接收的 Intent 类型。例如，通过为 Activity 声明 Intent 过滤器，可以使其他应用能够直接使用某一特定类型的 Intent 启动 Activity。如果没有为 Activity 声明任何 Intent 过滤器，则 Activity 只能通过显式 Intent 启动。

3.3 构建 Intent

Intent 对象携带了 Android 系统用来确定要启动哪个组件的信息（例如，准确的组件名称或应当接收该 Intent 的组件类别），以及接收信息的组件为了正确执行操作而使用的信息（例如，要采取的操作以及要处理的数据）。

Intent 中包含的主要信息如下。

1．组件名称

这是可选项，但也是构建显式 Intent 的一项重要信息，这意味着 Intent 应当仅传递给由组件名称定义的应用组件。

如果没有组件名称，则 Intent 是隐式的，且系统将根据其他 Intent 信息（例如，以下所述的操作、数据和类别）决定哪个组件应当接收 Intent。

因此，如需在应用中启动特定的组件，则应指定该组件的名称。

Intent 的这一字段是一个 ComponentName 对象，可以使用目标组件的完全限定类名指定此对象，其中包括应用的软件包名称，例如，com.example.ExampleActivity。

可以使用 setComponent()、setClass()、setClassName() 或 Intent 构造函数设置组件名称。

2．操作

操作是指要执行的通用操作（例如，“查看”或“发送”）的字符串，在很大程度上决定了 Intent 其余部分的构成，特别是数据和 Extra 中包含的内容。

可以创建自己的操作，但是，通常应该使用由 Intent 类或其他框架类定义的操作常量。以下是一些用于启动 Activity 的常见操作：

①ACTION_VIEW（"android.intent.action.VIEW"）

此操作用于查看信息（例如，要使用图库应用查看的照片；或者要使用浏览器应用打开网页），可以用 Intent 将此操作与 startActivity() 结合使用。

②ACTION_SEND（"android.intent.action.SEND"）

这也称为“分享”操作。可以用 Intent 将此操作与 startActivity()结合使用，分享数据。

③ACTION_EDIT（"android.intent.action.EDIT"）

此操作提供对给定数据的显式可编辑访问。

④ACTION_PICK（"android.intent.action.PICK"）

此操作从数据中选择一个项目，返回所选的内容。

⑤ACTION_DIAL（"android.intent.action.DIAL"）

此操作按指定的号码拨号，会显示一个界面，允许用户显式启动呼叫。

⑥ACTION_SENDTO（"android.intent.action.SENDTO"）

此操作给指定的号码发送消息。

⑦ACTION_DELETE（"android.intent.action.DELETE"）

此操作从容器中删除给定的数据。

⑧ACTION_WEB_SEARCH（"android.intent.action.WEB_SEARCH"）

此操作在网络上搜索内容。

3．数据

数据包括待操作数据和该数据 MIME 类型的 URI（Uri 对象）。提供的数据类型通常由 Intent 的操作决定。例如，如果操作是 ACTION_EDIT，则数据应包含待编辑文档的 URI。

创建 Intent 时，除了指定 URI 以外，指定数据类型（其 MIME 类型）往往也很重要。例如，能够显示图像的 Activity 可能无法播放音频文件，即便 URI 格式十分类似时也是如此。因此，指定数据的 MIME 类型有助于 Android 系统找到接收 Intent 的最佳组件。但有时 MIME 类型可以从 URI 中推断得出，特别当数据是 content: URI 时尤其如此。这表明数据位于设备中，且由 ContentProvider 控制，这使得数据 MIME 类型对系统可见。

要仅设置数据 URI，需调用 setData()。要仅设置 MIME 类型，则调用 setType()。如有必要，可以使用 setDataAndType() 同时显式设置两者。

注意：若要同时设置 URI 和 MIME 类型，请勿调用 setData() 和 setType()，因为它们会互相抵消彼此的值。务必始终使用 setDataAndType() 同时设置 URI 和 MIME 类型。

4．类别

类别是一个包含应处理 Intent 组件类型的附加信息的字符串。可以将任意数量的类别描述放入一个 Intent 中，但大多数 Intent 均不需要类别。

以下是一些常见类别：

```
①CATEGORY_HOME（"android.intent.category.HOME"）
```

此类别表明 Activity 是应用启动后，第一个被显示的 Activity。

```
②CATEGORY_LAUNCHER（"android.intent.category.LAUNCHER"）
```

此类别表明 Activity 是任务的初始 Activity，在系统的应用启动器中可以看到。可以使用 addCategory()指定类别。

以上列出的这些属性（组件名称、操作、数据和类别）表示 Intent 的既定特征，通过读取这些属性，Android 系统能够解析应当启动哪个应用组件。

Intent 还可以提供以下信息。

5．Extra

Extra 是携带完成请求操作所需的附加信息的键值对。正如某些操作使用特定类型的数据 URI 一样，有些操作也使用特定的 Extra。

可以使用各种 putExtra() 函数添加 Extra 数据，每个函数均接收两个参数：键名和值。还可以创建一个包含所有 Extra 数据的 Bundle 对象，然后使用 putExtras()将 Bundle 插入 Intent 中。

例如，使用 ACTION_SEND 创建用于发送电子邮件的 Intent 时，可以使用 EXTRA_EMAIL 键指定收件人，并使用 EXTRA_SUBJECT 键指定“主题”。

Intent 类为标准化的数据类型指定多个 EXTRA_* 常量。如需声明自己的 Extra 键（对于应用接收的 Intent），需确保将应用的软件包名称作为前缀。

例如：

```
static final String EXTRA_LOGIN = "com.example.EXTRA_LOGIN";
```

6．Flag

Flag 是在 Intent 类中定义的、充当 Intent 元数据的标志。

标志可以指示 Android 系统如何启动 Activity，以及启动之后如何处理，如：

```
FLAG_ACTIVITY_NEW_TASK
```

表明在新任务中启动 Activity。

```
FLAG_ACTIVITY_NO_HISTORY
```

表明新的 Activity 将在历史堆栈中不被保留。

3.4 显式 Intent 示例

显式 Intent 是指用于启动某个特定应用组件（例如，应用中的某个特定 Activity 或服务）的 Intent。要创建显式 Intent，需要为 Intent 对象定义组件名称，Intent 的所有其他属性均为可选属性。

例如，如果在应用中构建了一个名为 LogInActivity 的 Activity 用于登录功能，可以用以下代码启动此 Activity：

```
Intent intent = new Intent(this, LogInActivity.class);
startActivity(intent);
```

Intent(Context, Class) 构造函数的两个参数是 Activity 的 Context 和 Class 对象。

3.5 隐式 Intent 示例

如果当前的应用无法执行某项操作，而用户使用的设备上其他应用可以，且希望用户选取要使用的应用，那就可以使用隐式 Intent。

例如，如果希望用户与他人分享内容，可以使用 ACTION_SEND 操作创建 Intent，并添加指定共享内容的 Extra。使用该 Intent 调用 startActivity() 时，用户可以选取分享内容所使用的应用。

注意：*用户设备上可能没有任何应用能够处理当前应用发送到 startActivity() 的隐式 Intent。如果出现这种情况，则调用将会失败，且应用会崩溃。*

要验证 Activity 是否会接收 Intent，需要对 Intent 对象调用 resolveActivity()。如果结果为非空，则至少有一个应用能够处理该 Intent，且可以安全调用 startActivity()。如果结果为空，则不应使用该 Intent。如有可能，应停用发出该 Intent 的功能。

在 App Inventor 2 的联系人选择框组件的源码文件 ContactPicker.java 中，有如下代码：

```
@SimpleFunction(description = "view a contact via its URI")
public void ViewContact(String uri) {
  if(contactUri != null){
      Intent intent = new Intent(Intent.ACTION_VIEW,Uri.parse(uri));
      if (intent.resolveActivity(this.activityContext.getPackageManager()) != null) {
          this.activityContext.startActivity(intent);
      }
  }
}
```

其中就使用了 resolveActivity()函数判断是否有应用能处理 Intent。

在 App Inventor 2 的信息分享器组件的源码文件 Sharing.java 中，有如下代码：

```
@SimpleFunction(description = "Shares a message through any capable " +
```

```
            "application installed on the phone by displaying a list of the available apps and " +
            "allowing the user to choose one from the list. The selected app will open with the " +
            "message inserted on it.")
    public void ShareMessage(String message) {
        Intent shareIntent = new Intent(Intent.ACTION_SEND);
        shareIntent.putExtra(Intent.EXTRA_TEXT, message);
        shareIntent.setType("text/plain");

        // We cannot use Intent.createChooser(shareIntent, "Send using...") because it creates an
        // oversized pop up sharing window.
        this.form.startActivity(shareIntent);
    }
```

上述代码没有使用 URI，但声明了 Intent 的数据类型，用于指定 Extra 携带的内容。

调用 startActivity() 时，系统将检查已安装的所有应用，确定哪些应用能够处理这种 Intent。

如果只有一个应用能够处理，则该应用将立即运行并为其处理 Intent。如果有多个应用能够处理 Intent，则系统将显示一个对话框，使用户能够选取要使用的应用。如图 3-2 所示。



图 3-2　分享应用选择界面

用户还可以将选择的应用设置为该操作的默认选项，如果用户可能希望今后一直使用相

同的应用执行某项操作（例如，打开网页时，用户往往倾向于仅使用一种网络浏览器），则这一点十分有用。

如果用户希望每次使用不同的应用，则应采用显式方式显示选择器对话框。选择器对话框每次都会要求用户选择执行操作的应用。

3.6　Intent 过滤器

Intent 过滤器主要声明以下内容：

1）Intent 操作。

2）Intent 数据（URI 和数据类型）。

3）Intent 类别。

示例如下：

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(Intent.ACTION_VIEW);
try {
    intentFilter.addDataType("audio/mpeg");
} catch (IntentFilter.MalformedMimeTypeException e)
{
    e.printStackTrace();
}
intentFilter.addCategory(Intent.CATEGORY_DEFAULT);
```

Intent 过滤器主要有两个用途：

1）系统收到隐式 Intent 以启动 Activity 时，它根据操作、数据和类别将该 Intent 与 Intent 过滤器进行比较，搜索执行该 Intent 的最佳 Activity。

2）用 Intent 的操作、数据和类别内容，确定消息接收器要接收的 Intent 广播。

在 App Inventor 2 中，Intent 过滤器主要用于第 2 个用途。

如在电话拨号器组件的源码文件 PhoneCall.java 中，有如下代码：

```
private void registerCallStateMonitor(){
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
intentFilter.addAction(TelephonyManager.ACTION_PHONE_STATE_CHANGED);
    context.registerReceiver(callStateReceiver, intentFilter);
}
```

上述代码定制消息接收器接收 Intent.ACTION_NEW_OUTGOING_CALL 和 TelephonyManager.ACTION_PHONE_STATE_CHANGED 两个 Intent 广播。

第 4 章　Broadcast 介绍

4.1　Broadcast 机制概述

在 Android 系统中，要使用广播，需要有广播发送者和广播接收者，通常情况下，BroadcastReceiver 指的就是广播接收者（广播接收器）。

广播作为 Android 组件间的通信方式，可以使用的场景如下：

1）应用内部的消息通信。

2）不同应用之间的消息通信。

3）Android 系统在特定情况下与应用之间的消息通信。

App Inventor 2 系统中使用广播主要是在 Android 系统与应用间进行消息通信。

4.2　BroadcastReceiver

自定义广播接收器需要继承基类 BroadcastReceiver，并实现抽象函数 onReceive(context, intent)。广播接收器接收到相应广播后，会自动调用 onReceive()函数。

默认情况下，广播接收器也是运行在主线程，因此，onReceive 函数中不能执行太耗时的操作（不要超过 10s），否则将会产生 ANR（Application Not Responding）问题。

onReceive 函数中涉及与其他组件之间的交互，可以用发送 Notification、启动 Service 等方式，最好不要启动 Activity。

BroadcastReceiver 总体上可以分为两种注册方式：静态注册和动态注册。

4.2.1　静态注册

静态注册是指直接在 AndroidManifest.xml 文件中进行注册，涉及的相关属性如下：

```
<receiver android:directBootAware=["true" | "false"]
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

其中，以下两个属性需特别关注：

1）android:exported ——此 BroadcastReceiver 能否接收其他应用发出的广播，当设为 false 时，只能接收同一应用的组件，或具有相同 user ID 的应用发送的消息。

这个属性的默认值是由 receiver 中有无 intent-filter 决定的，如果有 intent-filter，默认值为 true，否则为 false。

2）android:permission ——如果设置，具有相应权限的广播发送方发送的广播才能被此 BroadcastReceiver 所接收；如果没有设置，属性的值就是整个应用所申请的权限。

常见的注册形式如下：

```
<receiver android:name=".EamBroadcastReceiver"   android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

4.2.2 动态注册

动态注册时，无须在 AndroidManifest.xml 文件中注册<receiver/>组件，直接在代码中通过调用 Context 的 registerReceiver 函数，可以动态注册 BroadcastReceiver。

在 App Inventor 2 中，采用的注册类型主要是动态注册。在电话拨号器组件的源码文件 PhoneCall.java 中，有如下代码：

```
//注册广播接收器
private void registerCallStateMonitor(){
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
    intentFilter.addAction(TelephonyManager.ACTION_PHONE_STATE_CHANGED);
    context.registerReceiver(callStateReceiver, intentFilter);
}

private void unregisterCallStateMonitor(){
    context.unregisterReceiver(callStateReceiver);
}
//销毁 Activity 时取消注册广播接收器;
@Override
public void onDestroy() {
    unregisterCallStateMonitor();
}
```

4.3 系统广播

Android 系统中内置了多个系统广播（System Broadcast），只要涉及手机的基本操作，

基本上都会发出相应的系统广播，如开机启动、网络状态改变和电量不足等。每个系统广播都具有特定的 intent-filter，其中主要包括具体的 action，系统广播发出后，将被相应的 BroadcastReceiver 接收。系统广播由系统内部在特定事件发生时自动发出。

以下是几个定义系统广播的字符串常量：

```
//设备进入低电量状态
ACTION_BATTERY_LOW（"android.intent.action.BATTERY_LOW"）

//用户打开或关闭飞行模式
ACTION_AIRPLANE_MODE_CHANGED（"android.intent.action.AIRPLANE_MODE"）

//网络连接发生了变化
CONNECTIVITY_ACTION（"android.net.conn.CONNECTIVITY_CHANGE"）
```

第 5 章 权 限 介 绍

5.1 权限简介

权限是 Android 系统内置的一种限制，用于限制对部分代码或设备上数据的访问。施加限制是为了保护可能被误用以致破坏或损害用户体验的关键数据和代码。

如果应用需要访问受权限保护的功能，则必须在清单文件中使用<uses-permission> 元素声明应用需要该权限。将应用安装到设备上时，安装程序会检查签署应用证书的颁发机构并（在某些情况下）询问用户确定是否授予请求的权限。如果授予权限，则应用能够使用受保护的功能；否则，其访问这些功能的尝试将会失败，并且不会向用户发送任何通知。

5.2 常用权限说明

每种权限均由一个字符串标识，指示受限制的操作。以下是 Android 系统定义的一些权限字符串：

```
//设置墙纸
android.permission.SET_WALLPAPER

//获取设备的联网状态
android.permission.ACCESS_NETWORK_STATE

//改变网络连接状态
android.permission.CHANGE_NETWORK_STATE

//获取设备的 WIFI 状态
android.permission.ACCESS_WIFI_STATE

//改变 WIFI 连接状态
android.permission.CHANGE_WIFI_STATE

//使用蓝牙功能
android.permission.BLUETOOTH
android.permission.BLUETOOTH_ADMIN

//使用闪光灯
android.permission.FLASHLIGHT
```

```
//连接网络
android.permission.INTERNET

//接收开机广播
android.permission.RECEIVE_BOOT_COMPLETED

//使用震动功能
android.permission.VIBRATE

//使用照相机功能
android.permission.CAMERA

//使用定位功能
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION

//使用录音功能
android.permission.RECORD_AUDIO

//读取手机状态，包括设备的电话号码、当前的蜂窝网络信息和正在进行的通话状态等
android.permission.READ_PHONE_STATE

//拨打电话
android.permission.CALL_PHONE

//处理接收到的电话
android.permission.ANSWER_PHONE_CALLS

//读写通话记录
android.permission.READ_CALL_LOG
android.permission.WRITE_CALL_LOG

//发送、接收和读取短信
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.READ_SMS

//接收 WAP PUSH 和彩信
android.permission.RECEIVE_WAP_PUSH
android.permission.RECEIVE_MMS

//读写存储空间
android.permission.READ_EXTERNAL_STORAGE
android.permission.WRITE_EXTERNAL_STORAGE
```

第6章　注 解 介 绍

注解就是代码的元数据（元数据是一种描述数据的数据），可以声明在类、变量、函数和函数参数等的前面，用来对这些元素进行说明。下面介绍 App Inventor 2 源码中使用到的一些注解。

6.1　DesignerComponent

此注解用于标记组件设计时使用的组件，源码文件为 DesignerComponent.java，代码如下：

```
package com.google.appinventor.components.annotations;

import com.google.appinventor.components.common.ComponentCategory;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

//注解会被保存在 class 文件中，在运行时可以通过反射获取到
@Retention(RetentionPolicy.RUNTIME)

//注解的作用目标是类或接口
@Target(ElementType.TYPE)
public @interface DesignerComponent {
   ComponentCategory category() default ComponentCategory.UNINITIALIZED;

   String description() default "";

   String designerHelpDescription() default "";

   boolean showOnPalette() default true;

   boolean nonVisible() default false;

   String iconName() default "";

   int version();
```

```
        String helpUrl() default "";
    }
```

Category：组件的类型，可取的值如下：

```
    public enum ComponentCategory {
        USERINTERFACE("User Interface"),
        LAYOUT("Layout"),
        MEDIA("Media"),
        ANIMATION("Drawing and Animation"),
        SENSORS("Sensors"),
        SOCIAL("Social"),
        STORAGE("Storage"),
        CONNECTIVITY("Connectivity"),
        LEGOMINDSTORMS("LEGO\u00AE MINDSTORMS\u00AE"),
        EXPERIMENTAL("Experimental"),
        EXTENSION("Extension"),
        INTERNAL("For internal use only"),
        // UNINITIALIZED is used as a default value so Swing libraries can still compile
    UNINITIALIZED("Uninitialized");
    …
    }
```

description 和 designerHelpDescription 的内容是关于此组件的一些说明文字，通常只需给 description 赋值就可以了。

在组件面板界面，单击某个组件右边的问号图标，就会显示 description 的内容，如图 6-1 所示。



图 6-1　组件 description 内容显示界面

showOnPalette：是否在组件面板中显示此组件，默认为 true。

nonVisible：是否在 UI 界面上显示此组件，即是否是非可视组件，默认为 false。

iconName：组件的图标文件名称。

version：组件的版本号。

helpUrl：组件帮助文档的 URL 地址，主要是给插件用的。

6.2 SimpleObject

此注解用于标记对象，源码文件为 SimpleObject.java，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface SimpleObject {

    boolean external() default false;

}
```

external：表示组件是否是外部组件，默认值为 false。如果设为 true，会根据组件的 java 文件生成 aix 文件，主要用于插件开发。

6.3 UsesPermissions

此注解用于声明组件用到的 Android 系统权限，源码文件为 UsesPermissions.java，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface UsesPermissions {

  String permissionNames() default "";
}
```

permissionNames：Android 系统权限名称字符串，一次可声明多个权限，中间用逗号隔开。

6.4 DesignerProperty

此注解用于声明组件的属性在组件属性面板中可见，组件属性的 setter 函数必须用此注解声明，源码文件为 DesignerProperty.java，代码如下：

```
package com.google.appinventor.components.annotations;

import com.google.appinventor.components.common.PropertyTypeConstants;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

//此注解的作用目标是方法，也就是函数
@Target(ElementType.METHOD)
public @interface DesignerProperty {

  String editorType() default PropertyTypeConstants.PROPERTY_TYPE_TEXT;

  String defaultValue() default "";
}
```

editorType：属性的参数类型，常用的类型有以下几个。

```
public class PropertyTypeConstants {
  …
  public static final String PROPERTY_TYPE_BOOLEAN = "boolean";

  public static final String PROPERTY_TYPE_FLOAT = "float";

  public static final String PROPERTY_TYPE_INTEGER = "integer";

  public static final String PROPERTY_TYPE_NON_NEGATIVE_FLOAT = "non_negative_float";

  public static final String PROPERTY_TYPE_NON_NEGATIVE_INTEGER = "non_negative_
  integer";

  //这两个类型是一样的，优先使用 PROPERTY_TYPE_STRING
  public static final String PROPERTY_TYPE_STRING = "string";
  public static final String PROPERTY_TYPE_TEXT = "text";

  public static final String PROPERTY_TYPE_VISIBILITY = "visibility";
```

```
    …
}
```

6.5 SimpleProperty

此注解用于声明组件的属性，属性的 getter 和 setter 函数都必须用此注解声明，源码文件为 SimpleProperty.java，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface SimpleProperty {

    String description() default "";

    PropertyCategory category() default PropertyCategory.UNSET;

    boolean userVisible() default true;
}
```

description 的内容为属性的说明文字。

在工作面板界面，当把鼠标移动到设置或获取属性值的代码块上时，就会显示 description 的内容，如图 6-2 所示。



图 6-2　属性的 description 内容显示界面

category：属性的类型，该变量要么用于 getter 函数，要么用于 setter 函数，不需要两个函数都使用。可取的值如下：

```
public enum PropertyCategory {
    BEHAVIOR("Behavior"),
```

```
    APPEARANCE("Appearance"),
    DEPRECATED("Deprecated"),
    UNSET("Unspecified");
  …
}
```

userVisible：设置属性是否能通过代码块访问，默认为 true。

6.6 SimpleFunction

此注解用于声明组件的功能函数，源码文件为 SimpleFunction.java，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface SimpleFunction {

  String description() default "";

  boolean userVisible() default true;
}
```

description 的内容为此函数的说明文字。

在工作面板界面，当把鼠标移动到函数的代码块上时，就会显示 description 的内容，如图 6-3 所示。



图 6-3 功能函数的 description 内容显示界面

userVisible：设置能否通过代码块访问此函数，默认为 true。

6.7 SimpleEvent

此注解用于声明组件的事件响应函数，源码文件为 SimpleEvent.java，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface SimpleEvent {

  String description() default "";

  boolean userVisible() default true;
}
```

description 的内容为此函数的说明文字。

在工作面板界面，当把鼠标移动到函数的代码块上时，就会显示 description 的内容，如图 6-4 所示。



图 6-4　事件响应函数的 description 内容显示界面

userVisible：设置能否通过代码块访问此函数，默认为 true。

6.8　UsesBroadcastReceivers

此注解用于声明组件使用的广播接收器，源码文件为 UsesBroadcastReceivers.java，代码如下：

```
package com.google.appinventor.components.annotations;

import com.google.appinventor.components.annotations.androidmanifest.ReceiverElement;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface UsesBroadcastReceivers {
   ReceiverElement[] receivers();
}
```

receivers：接收器数组。

6.9 ReceiverElement

此注解用于声明组件使用的广播接收器的各属性，源码文件为 ReceiverElement，代码如下：

```
package com.google.appinventor.components.annotations.androidmanifest;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ReceiverElement {

   IntentFilterElement[] intentFilters() default {};

   MetaDataElement[] metaDataElements() default {};

   String name();

   String enabled() default "";

   String exported() default "";

   String icon() default "";

   String label() default "";

   String permission() default "";

   String process() default "";
}
```

intentFilters：Intent 过滤器数组。
metaDataElements：元数据数组。
name：广播接收器的类名。

enabled：广播接收器能否被系统实例化。
exported：广播接收器能否接收从本应用之外发来的消息。
icon：广播接收器的图标。
label：广播接收器的文字标签。
permission：给广播接收器发送消息的发送方必须拥有的权限名称。
process：广播接收器所属进程的名称。

6.10 IntentFilterElement

此注解用于声明广播接收器的 Intent 过滤器属性，源码文件为 IntentFilterElement.java，代码如下：

```
package com.google.appinventor.components.annotations.androidmanifest;

public @interface IntentFilterElement {

    ActionElement[] actionElements();

    CategoryElement[] categoryElements() default {};

    DataElement[] dataElements() default {};

    String icon() default "";

    String label() default "";

    String priority() default "";

}
```

actionElements：Intent 操作数组。
categoryElements：Intent 类别数组。
dataElements：Intent 类别数组。
icon 和 label：Intent 过滤器的图标和文字标签。
priority：广播接收器处理此 Intent 过滤器确定的 Intent 广播的优先级。

6.11 ActionElement

此注解用于声明 Intent 操作名称，源码文件为 ActionElement.java，代码如下：

```
package com.google.appinventor.components.annotations.androidmanifest;

import java.lang.annotation.ElementType;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ActionElement {

    String name();

}
```

name：操作的名称。

6.12 UsesLibraries

此注解用于声明使用到的库文件，源码文件为 UsesLibraries，代码如下：

```
package com.google.appinventor.components.annotations;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Annotation to indicate library files required by components.
 *
 * @author ralph.morelli@trincoll.edu
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface UsesLibraries {

    /**
     * The names of the libraries separated by commas.
     *
     * @return    the library name
     * @see android.Manifest.permission
     */
    String libraries() default "";
}
```

libraries：库文件名称。

第 7 章　Screen 的定制

Screen 是 App Inventor 2 开发使用的最基础组件，是一个容器框架，其余组件都附着在这个组件上，其对应的源码文件为

```
/appinventor/components/src/com/google/appinventor/components/ runtime/Form.java
```

7.1　功能菜单的修改

使用 App Inventor 2 开发的应用，每个 Screen 都默认有两个功能菜单：“Stop this application”和“About this application”，显示的字符串都是英文，且常用的“设置”和“更新版本”菜单也没有，用户体验不好。

现修改 Form.java 的代码，修改原有菜单显示的字符串为中文和增加功能菜单，具体如下。

7.1.1　原有菜单的修改

“Stop this application”和“About this application”两个菜单使用的字符串都是英文，为方便使用，修改为中文。代码如下：

```
//“Stop this application”菜单函数
public void addExitButtonToMenu(Menu menu) {
    //1 表示此菜单在菜单列表中排在第 1 位，就是索引值为 1
    MenuItem stopApplicationItem = menu.add(Menu.NONE, Menu.NONE, 1, //Menu.FIRST,
    //"Stop this application")
    "退出")
    .setOnMenuItemClickListener(new OnMenuItemClickListener() {
      public boolean onMenuItemClick(MenuItem item) {
        //单击菜单调用的函数
        showExitApplicationNotification();
        return true;
      }
});
    //菜单图标
    stopApplicationItem.setIcon(android.R.drawable.ic_menu_close_clear_cancel);
  }

  //“About this application”菜单函数
```

```
    public void addAboutInfoToMenu(Menu menu) {
        MenuItem aboutAppItem = menu.add(Menu.NONE, Menu.NONE, 2, //2,
        //"About this application")
        "关于")
        .setOnMenuItemClickListener(new OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {
                showAboutApplicationNotification();
                return true;
            }
        });
        aboutAppItem.setIcon(android.R.drawable.ic_menu_info_details);
    }

  private void showExitApplicationNotification() {
        //String title = "Stop application?";
        //String message = "Stop this application and exit? You'll need to relaunch " +
        //       "the application to use it again.";
        //String positiveButton = "Stop and exit";
        //String negativeButton = "Don't stop";
        String title = "提示";
        String message = "确定要退出应用？ ";
        String positiveButton = "确定";
        String negativeButton = "取消";
        // These runnables are passed to twoButtonAlert.    They perform the corresponding actions
        // when the button is pressed.    Here there's nothing to do for "don't stop" and cancel
        Runnable stopApplication = new Runnable() {public void run () {closeApplicationFromMenu();}};
        Runnable doNothing = new Runnable () {public void run() {}};
        Notifier.twoButtonDialog(
                this,
                message,
                title,
                positiveButton,
                negativeButton,
                false, // cancelable is false
                stopApplication,
                doNothing,
                doNothing);
    }

    private void showAboutApplicationNotification() {
        //String title = "About this app";
        String title = "关于";

        String MITtagline = "<p><small><em>Invented with MIT App Inventor<br>appinventor.mit.edu
        </em></small></p>";
```

```
        // Users can hide the taglines by including an HTML open comment <!-- in the about screen message
        //String message = aboutScreen + MITtagline + yandexTranslateTagline;
        String message = aboutScreen;

        message = message.replaceAll("\\n", "<br>"); // Allow for line breaks in the string.
        //String buttonText ="Got it";
        String buttonText ="确定";

        Notifier.oneButtonAlert(this, message, title, buttonText);
    }
```

7.1.2 增加设置新增菜单的属性

为了使开发人员能够比较灵活地使用“设置”和“更新版本”菜单，增加两个属性以动态设置是否增加这两个菜单。代码如下：

```
//增加变量标识是否显示“设置”和“更新版本”菜单
private boolean isSettingMenu;
private boolean isUpdateMenu;

//增加属性的获取和设置函数
@SimpleProperty(category = PropertyCategory.APPEARANCE,
        description = "设置是否显示设置菜单")
public boolean IsSettingMenu() {
    return isSettingMenu;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
    defaultValue = "False")
@SimpleProperty
public void IsSettingMenu(boolean isSettingMenu) {
        this.isSettingMenu = isSettingMenu;
}

@SimpleProperty(category = PropertyCategory.APPEARANCE,
        description = "设置是否显示更新 App 菜单")
public boolean IsUpdateMenu() {
    return isUpdateMenu;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
    defaultValue = "False")
@SimpleProperty
public void IsUpdateMenu(boolean isUpdateMenu) {
        this.isUpdateMenu = isUpdateMenu;
}
```

定义好属性后，还需要在 OdeMessages.java 中添加属性的声明：

```
@DefaultMessage("IsSettingMenu")
@Description("")
String IsSettingMenuProperties();

@DefaultMessage("IsUpdateMenu")
@Description("")
String IsUpdateMenuProperties();
```

到目前为止，已经成功添加了属性，可以正常使用了，但属性名称在中文环境下也显示为英文，需要在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
IsSettingMenuProperties = 是否显示设置菜单
IsUpdateMenuProperties = 是否显示更新版本菜单
```

7.1.3 新增菜单的实现

```
//创建选项菜单时，根据 isSettingMenu 和 isUpdateMenu 的值，确定是否新增菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // This procedure is called only once.   To change the items dynamically
  // we would use onPrepareOptionsMenu.
  super.onCreateOptionsMenu(menu);
  // add the menu items
  // Comment out the next line if we don't want the exit button
  addExitButtonToMenu(menu);
  addAboutInfoToMenu(menu);

  if (isSettingMenu) {
    addSettingToMenu(menu);
  }

  if (isUpdateMenu) {
    addUpdateToMenu(menu);
  }

  for (OnCreateOptionsMenuListener onCreateOptionsMenuListener : onCreateOptionsMenuListeners) {
    onCreateOptionsMenuListener.onCreateOptionsMenu(menu);
  }
  return true;
}

//“设置”菜单
public void addSettingToMenu(Menu menu) {
    MenuItem updateAppItem = menu.add(Menu.NONE, Menu.NONE, 3,
```

```
            //"About this application")
            "设置")
            .setOnMenuItemClickListener(new OnMenuItemClickListener() {
                public boolean onMenuItemClick(MenuItem item) {
                    Setting();
                    return true;
                }
            });
            updateAppItem.setIcon(android.R.drawable.ic_menu_preferences);
        }

        // “更新版本”菜单
        public void addUpdateToMenu(Menu menu) {
            if (!isSettingMenu) {
                MenuItem updateAppItem = menu.add(Menu.NONE, Menu.NONE, 3,
                //"About this application")
                "更新版本")
                .setOnMenuItemClickListener(new OnMenuItemClickListener() {
                    public boolean onMenuItemClick(MenuItem item) {
                        Update();
                        return true;
                    }
                });
                updateAppItem.setIcon(android.R.drawable.ic_menu_more);
            } else {
                MenuItem updateAppItem = menu.add(Menu.NONE, Menu.NONE, 4,
                //"About this application")
                "更新版本")
                .setOnMenuItemClickListener(new OnMenuItemClickListener() {
                    public boolean onMenuItemClick(MenuItem item) {
                        Update();
                        return true;
                    }
                });
                updateAppItem.setIcon(android.R.drawable.ic_menu_more);
            }
        }
```

因为不确定开发人员具体要实现怎样的设置功能和更新版本功能，现提供两个事件响应函数供开发人员调用：

```
    @SimpleEvent(description = "单击设置菜单的响应函数")
    public void Setting() {
        EventDispatcher.dispatchEvent(this, "Setting");
    }

    @SimpleEvent(description = "单击更新版本菜单的响应函数")
```

```
public void Update() {
    EventDispatcher.dispatchEvent(this, "Update");
}
```

定义好函数后，还需要在 OdeMessages.java 中添加函数的声明：

```
@DefaultMessage("Setting")
@Description("")
String SettingEvents();

@DefaultMessage("Update")
@Description("")
String UpdateEvents();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
SettingEvents = 单击设置菜单的响应函数
UpdateEvents = 单击更新版本菜单的响应函数
```

编译、运行系统后，在组件属性中，可以看到多了“是否显示设置菜单”和“是否显示更新版本菜单”两个属性，默认为不显示，如图 7-1 所示。

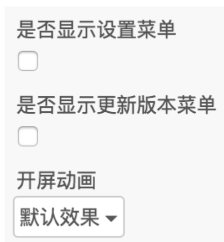


图 7-1　新增属性

在工作面板界面，可以看到多了“单击设置菜单的响应函数”和“单击更新版本菜单的响应函数”两个事件响应函数，以及多了“是否显示设置”菜单和“是否显示更新版本”菜单两个属性的 getter 和 setter 函数，如图 7-2 和图 7-3 所示。



图 7-2　新增事件响应函数



图 7-3　新增属性函数

当选中显示“设置”菜单和“更新版本”菜单，编译并运行应用后，单击设备上的菜单键，在应用界面的底部会显示 4 个中文功能菜单，如图 7-4 所示。

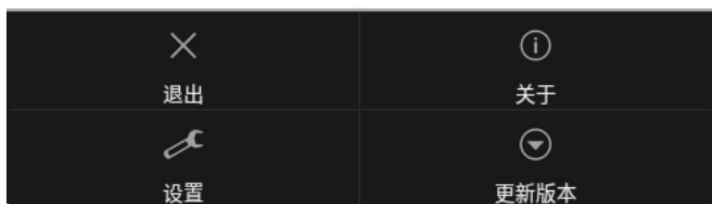


图 7-4　功能菜单

单击“退出”功能菜单，显示中文提示框，如图 7-5 所示。



图 7-5　“退出”功能菜单

“单击设置菜单的响应函数”和“单击更新版本菜单的响应函数”两个事件响应函数使用示例代码如图 7-6 和图 7-7 所示。

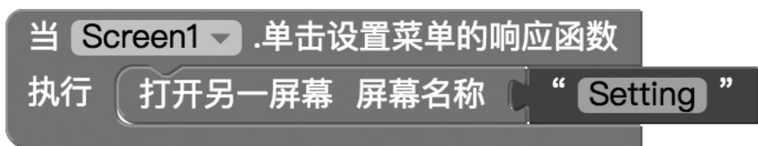


图 7-6　代码 1



图 7-7　代码 2

在单击“设置”菜单的时候，跳转到设置界面；单击“更新版本”菜单的时候，跳转到一个展示新版应用信息的网页，可以下载新版本。

7.2　增加暂停和恢复状态事件响应函数

从之前 Activity 介绍章节可知，当 Activity 进入到暂停状态后，系统可能会回收当前应用使用的内存，这样当 Activity 再次进入到恢复状态后，应用的数据可能会丢失。而且

Activity 进入到暂停状态后，用户无法再看到当前屏幕，需要停止播放动画、视频等，当Activity 进入到恢复状态后，需要重新播放动画、视频等。

在 App Inventor 2 中，Form 就对应 Activity，可以在 Form 中增加两个事件函数："当前屏幕进入暂停状态"和"当前屏幕恢复运行状态"，开发人员可以利用前者，保存数据、停止播放动画和视频等；利用后者，读取保存的数据、重新播放动画和视频等。

实现方式如下：

```
//增加当前屏幕进入暂停状态的事件函数
@Override
protected void onPause() {
    super.onPause();
    Log.i(LOG_TAG, "Form " + formName + " got onPause");
    for (OnPauseListener onPauseListener : onPauseListeners) {
        onPauseListener.onPause();
    }

    ScreenPause();
}

@SimpleEvent(description = "当前屏幕进入暂停状态")
public void ScreenPause() {
    EventDispatcher.dispatchEvent(this, "ScreenPause");
}

//增加当前屏幕进入运行状态的事件函数
@Override
protected void onResume() {
    super.onResume();
    Log.i(LOG_TAG, "Form " + formName + " got onResume");
    activeForm = this;

    // If applicationIsBeingClosed is true, call closeApplication() immediately to continue
    // unwinding through all forms of a multi-screen application.
    if (applicationIsBeingClosed) {
        closeApplication();
        return;
    }

    for (OnResumeListener onResumeListener : onResumeListeners) {
        onResumeListener.onResume();
}

    ScreenResume();
}
@SimpleEvent(description = "当前屏幕进行运行状态")
public void ScreenResume() {
```

```
        EventDispatcher.dispatchEvent(this, "ScreenResume");
    }
```

在 OdeMessages.java 中添加函数的声明：

```
    @DefaultMessage("ScreenPause")
    @Description("")
    String ScreenPauseEvents();

    @DefaultMessage("ScreenResume")
    @Description("")
    String ScreenResumeEvents();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
    ScreenPauseEvents = 屏幕进入暂停状态
    ScreenResumeEvents = 屏幕进入运行状态
```

编译、运行系统后，在工作面板界面，可以看到多了“屏幕进入暂停状态”和“屏幕进入运行状态”两个事件响应函数，如图 7-8 所示。



图 7-8　新增事件响应函数

使用这两个函数的示例代码如图 7-9 所示。



图 7-9　代码 3

第 8 章　图像组件（Image）的定制

8.1　增加支持的动画形式

App Inventor 2 提供的图像组件只支持左右滑动的动画形式，当把鼠标移动到“动画形式”属性的代码块上时，可看到对于此属性的说明，如图 8-1 所示。



图 8-1　“动画形式”属性说明

现修改代码使其也支持上下滑动的动画形式。

图像组件的源码文件为 Image.java，其中有如下代码：

```
@SimpleProperty(description = "This is a limited form of animation that can attach " +
      "a small number of motion types to images.   The allowable motions are " +
      "ScrollRightSlow, ScrollRight, ScrollRightFast, ScrollLeftSlow, ScrollLeft, " +
      "ScrollLeftFast, and Stop",
      category = PropertyCategory.APPEARANCE)
// TODO(user): This should be changed from a property to an "animate" method, and have the choices
// placed in a dropdown.   Aternatively the whole thing should be removed and we should do
// something that is more consistent with sprites.
public void Animation(String animation) {
   AnimationUtil.ApplyAnimation(view, animation);
}
```

可以看到其实是调用 AnimationUtil.ApplyAnimation 函数实现动画，其中只有让图片左右滑动的代码：

```
public static void ApplyAnimation(View view, String animation) {
   // TODO(user): These string constants need to be extracted and defined somewhere else!
   // TODO(user): Also, the endless else-if is inefficient
   if (animation.equals("ScrollRightSlow")) {
      ApplyHorizontalScrollAnimation(view, false, 8000);
   } else if (animation.equals("ScrollRight")) {
```

```
            ApplyHorizontalScrollAnimation(view, false, 4000);
        } else if (animation.equals("ScrollRightFast")) {
            ApplyHorizontalScrollAnimation(view, false, 1000);
        } else if (animation.equals("ScrollLeftSlow")) {
            ApplyHorizontalScrollAnimation(view, true, 8000);
        } else if (animation.equals("ScrollLeft")) {
            ApplyHorizontalScrollAnimation(view, true, 4000);
        } else if (animation.equals("ScrollLeftFast")) {
            ApplyHorizontalScrollAnimation(view, true, 1000);
        } else if (animation.equals("Stop")) {
            view.clearAnimation();
        }
    }

    private static void ApplyHorizontalScrollAnimation(View view, boolean left, int speed) {
        float sign = left ? 1f : -1f;
        AnimationSet animationSet = new AnimationSet(true);
        animationSet.setRepeatCount(Animation.INFINITE);
        animationSet.setRepeatMode(Animation.RESTART);

        TranslateAnimation move = new TranslateAnimation(Animation.RELATIVE_TO_PARENT, sign * 0.70f,
                Animation.RELATIVE_TO_PARENT, sign * -0.70f, Animation.RELATIVE_TO_PARENT, 0,
                Animation.RELATIVE_TO_PARENT, 0);
        move.setStartOffset(0);
        move.setDuration(speed);
        move.setFillAfter(true);
        animationSet.addAnimation(move);
        view.startAnimation(animationSet);
    }
```

参照源码添加让图片上下滑动的代码，首先修改 Image.java 中动画形式属性的说明文字，改为如下：

```
description = "设置图片左右或上下滑动，及停止滑动，可设置的值为：" +
        "ScrollRightSlow、ScrollRight、ScrollRightFast、ScrollLeftSlow、ScrollLeft、" +
"ScrollLeftFast、ScrollDownSlow、ScrollDown、ScrollDownFast、ScrollUpSlow、ScrollUp、
ScrollUpFast 和 Stop",
```

然后在 AnimationUtil.java 中添加让图片上下滑动的代码：

```
    public static void ApplyAnimation(View view, String animation) {
        // TODO(user): These string constants need to be extracted and defined somewhere else!
        // TODO(user): Also, the endless else-if is inefficient
        if (animation.equals("ScrollRightSlow")) {
            ApplyHorizontalScrollAnimation(view, false, 8000);
        } else if (animation.equals("ScrollRight")) {
            ApplyHorizontalScrollAnimation(view, false, 4000);
```

```
        } else if (animation.equals("ScrollRightFast")) {
            ApplyHorizontalScrollAnimation(view, false, 1000);
        } else if (animation.equals("ScrollLeftSlow")) {
            ApplyHorizontalScrollAnimation(view, true, 8000);
        } else if (animation.equals("ScrollLeft")) {
            ApplyHorizontalScrollAnimation(view, true, 4000);
        } else if (animation.equals("ScrollLeftFast")) {
            ApplyHorizontalScrollAnimation(view, true, 1000);
        } else if (animation.equals("ScrollDownSlow")) {
            ApplyVerticalScrollAnimation(view, false, 8000);
        } else if (animation.equals("ScrollDown")) {
            ApplyVerticalScrollAnimation(view, false, 4000);
        } else if (animation.equals("ScrollDownFast")) {
            ApplyVerticalScrollAnimation(view, false, 1000);
        } else if (animation.equals("ScrollUpSlow")) {
            ApplyVerticalScrollAnimation(view, true, 8000);
        } else if (animation.equals("ScrollUp")) {
            ApplyVerticalScrollAnimation(view, true, 4000);
        } else if (animation.equals("ScrollUpFast")) {
            ApplyVerticalScrollAnimation(view, true, 1000);
        } else if (animation.equals("Stop")) {
            view.clearAnimation();
        }
    }

    private static void ApplyVerticalScrollAnimation(View view, boolean up, int speed) {

        AnimationSet animationSet = new AnimationSet(true);
        animationSet.setRepeatCount(Animation.INFINITE);
        animationSet.setRepeatMode(Animation.RESTART);

        TranslateAnimation move;

        if (up){
            move = new TranslateAnimation(Animation.RELATIVE_TO_SELF, 0.0f,
                        Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF,
                        1.0f, Animation.RELATIVE_TO_SELF, 0.0f);
        }else{
            move = new TranslateAnimation(Animation.RELATIVE_TO_SELF, 0.0f,
                        Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF,
                        0.0f, Animation.RELATIVE_TO_SELF, 1.0f);
        }
}
```

编译、运行系统后，在工作面板中，当鼠标移动到“动画形式”属性的代码块上时，可以看到修改后的说明文字，如图 8-2 所示。

设置 图像1 . 动画形式 为

设置图片左右或上下滑动，及停止滑动，可设置的值为： ScrollRightSlow、ScrollRight、ScrollRightFast、ScrollLeftSlow、ScrollLeft、ScrollLeftFast、ScrollDownSlow、ScrollDown、ScrollDownFast、ScrollUpSlow、ScrollUp、ScrollUpFast和Stop

图 8-2 “动画形式”属性新的说明

使用这个属性的示例代码如图 8-3 所示。



图 8-3 代码

8.2 增加设置动画播放次数和时间的功能

App Inventor 2 默认动画是无限循环播放，但系统自带的代码有缺陷，设置代码不起作用，动画播放只会播放一次，需要修改代码。修改如下：

```
private static void ApplyHorizontalScrollAnimation(View view, boolean left, int speed) {
    float sign = left ? 1f : -1f;
AnimationSet animationSet = new AnimationSet(true);
    //原有设置代码
    //animationSet.setRepeatCount(Animation.INFINITE);
    //animationSet.setRepeatMode(Animation.RESTART);

    TranslateAnimation move = new TranslateAnimation(Animation.RELATIVE_TO_PARENT, sign *
        0.70f,
        Animation.RELATIVE_TO_PARENT, sign * -0.70f, Animation.RELATIVE_TO_PARENT, 0,
        Animation.RELATIVE_TO_PARENT, 0);
    move.setStartOffset(0);
    move.setDuration(speed);
    move.setFillAfter(true);
    //需要在此设置播放次数和模式
    move.setRepeatCount(Animation.INFINITE);
    move.setRepeatMode(Animation.RESTART);
    animationSet.addAnimation(move);
    view.startAnimation(animationSet);
  }
```

修改后，动画可无限循环播放，但不能设置播放次数和时间，不够灵活。

现增加设置播放次数和时间的代码，首先需要修改 Image.java 的代码，修改如下：

```
//增加两个变量记录播放次数和播放时间
private int repeatCount = 0;
private int duration = 0;

//以下两个函数是播放次数属性的 getter 和 setter 函数，默认值为 0，此时用系统默认的播放次数
@SimpleProperty(category = PropertyCategory.APPEARANCE, description = "设置动画播放的次数")
public int RepeatCount() {
    return repeatCount;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
    defaultValue = "0")
@SimpleProperty
public void RepeatCount(int repeatCount) {
    this.repeatCount = repeatCount;
    //播放次数的值传递给播放动画的类
    AnimationUtil.ApplyAnimationRepeatCount(repeatCount);
}

//以下两个函数是播放时间属性的 getter 和 setter 函数，默认值为 0，此时用系统默认的播放时间
@SimpleProperty(category = PropertyCategory.APPEARANCE,
    description = "设置动画播放时间，单位是毫秒")
public int Duration() {
    return duration;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
      defaultValue = "0")
@SimpleProperty
public void Duration(int duration) {
    this.duration = duration;
    //播放时间的值传递给播放动画的类
    AnimationUtil.ApplyAnimationDuration(duration);
}
```

AnimationUtil.java 的代码修改如下：

```
//增加两个变量记录播放次数和播放时间
private static int mRepeatCount = 0;
private static int mDuration = 0;

//增加两个变量的 setter 函数，供 Image 类调用
public static void ApplyAnimationRepeatCount(int repeatCount) {
     mRepeatCount = repeatCount;
}
```

```
public static void ApplyAnimationDuration(int duration) {
        mDuration = duration;
}

private static void ApplyHorizontalScrollAnimation(View view, boolean left, int speed) {
        float sign = left ? 1f : -1f;
        AnimationSet animationSet = new AnimationSet(true);
        //animationSet.setRepeatCount(Animation.INFINITE);
        //animationSet.setRepeatMode(Animation.RESTART);

        TranslateAnimation move = new TranslateAnimation(Animation.RELATIVE_TO_PARENT, sign*0.70f,
                Animation.RELATIVE_TO_PARENT, sign * -0.70f, Animation.RELATIVE_TO_PARENT, 0,
                Animation.RELATIVE_TO_PARENT, 0);
        move.setStartOffset(0);

        //值为 0 时用系统默认的播放时间
        if (mDuration == 0){
                move.setDuration(speed);
        } else {
                move.setDuration(mDuration);
        }

        move.setFillAfter(true);

        //值为 0 时用系统默认的播放次数
        if (mRepeatCount == 0) {
                move.setRepeatCount(Animation.INFINITE);
          } else {
                move.setRepeatCount(mRepeatCount);
        }

        move.setRepeatMode(Animation.RESTART);
        animationSet.addAnimation(move);
        view.startAnimation(animationSet);
    }

    private static void ApplyVerticalScrollAnimation(View view, boolean up, int speed) {

        AnimationSet animationSet = new AnimationSet(true);
        //animationSet.setRepeatCount(Animation.INFINITE);
        //animationSet.setRepeatMode(Animation.RESTART);

        TranslateAnimation move;

        if (up){
                move = new TranslateAnimation(Animation.RELATIVE_TO_SELF, 0.0f,
```

```
                    Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF,
                    1.0f, Animation.RELATIVE_TO_SELF, 0.0f);
        }else{
            move = new TranslateAnimation(Animation.RELATIVE_TO_SELF, 0.0f,
                    Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF,
                    0.0f, Animation.RELATIVE_TO_SELF, 1.0f);
        }

        move.setStartOffset(0);

        if (mDuration == 0){
            move.setDuration(speed);
        } else {
            move.setDuration(mDuration);
        }

        move.setFillAfter(true);

        if (mRepeatCount == 0) {
            move.setRepeatCount(Animation.INFINITE);
         } else {
            move.setRepeatCount(mRepeatCount);
        }

        move.setRepeatMode(Animation.RESTART);
        animationSet.addAnimation(move);
        view.startAnimation(animationSet);
      }
```

在 OdeMessages.java 中添加属性的声明：

```
    @DefaultMessage("RepeatCount")
    @Description("")
    String RepeatCountProperties();

    @DefaultMessage("Duration")
    @Description("")
    String DurationProperties();
```

在 OdeMessages_zh_CN.properties 中添加属性的中文字符串：

```
    RepeatCountProperties = 动画播放次数
    DurationProperties = 动画播放时间
```

编译、运行系统后，在组件属性中，可以看到多了“动画播放时间”和“动画播放次数”两个属性，默认值都是 0，如图 8-4 所示。

在工作面板中，也可以看到增加了“动画播放时间”和“动画播放次数”两个属性的

getter 和 setter 函数，如图 8-5 所示。



图 8-4　新增属性



图 8-5　新增属性函数

8.3　增加单击事件响应函数

Android 系统的图像显示控件是可以响应用户单击事件的，但 App Inventor 2 提供的图像组件不支持此功能，使用多有不便，现增加此功能，具体增加的代码如下：

```
import com.google.appinventor.components.annotations.SimpleEvent;
import android.view.View.OnClickListener;

//Image 类增加对 OnClickListener 接口的继承
public final class Image extends AndroidViewComponent implements OnClickListener {

  //构造函数中注册监听单击事件函数
  public Image(ComponentContainer container) {
        super(container);

        …
        view.setOnClickListener(this);
  }

  //重写从 OnClickListener 接口继承的函数
  @Override
```

```
        public void onClick(View view) {
            Click();
        }

        @SimpleEvent(description = "用户单击图片事件的响应函数")
        public void Click() {
            EventDispatcher.dispatchEvent(this, "Click");
        }
    }
```

App Inventor 2 的源码中已经在 OdeMessages.java 中添加了 Click 函数的声明，在 OdeMessages_zh_CN.properties 中添加了 Click 函数的中文字符串，因此无须修改这两个文件。

编译、运行系统后，在工作面板中，可以看到增加了一个新函数，如图 8-6 所示。



图 8-6　新增事件响应函数

第 9 章　标签（Label）的定制

标签组件对应的是 Android 系统的 TextView 控件，App Inventor 2 默认只用于显示文本，但也可以扩展支持显示图片，源码文件为 Label.java。

9.1　增加允许单行或多行显示的属性

App Inventor 2 的标签组件默认支持文本长度大于组件长度时为多行显示，但有时从界面设计方面考虑，不能多行显示，只能单行显示；在单行显示的时候，如果文本长度大于组件长度，为了使用户能看到所有的文本，还需要支持文本的水平滚动，具体实现方式如下。

在 Label.java 中增加如下代码：

```
//增加变量标识单行或多行显示
private boolean multiLine;

@SimpleProperty( category = PropertyCategory.BEHAVIOR, description = "允许多行")
public boolean MultiLine() {
    return multiLine;
}

@DesignerProperty(editorType=PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue="True")
@SimpleProperty
public void MultiLine(boolean multiLine) {
    this.multiLine = multiLine;
    TextViewUtil.setSingleLine(view, !multiLine);
}
```

在 TextViewUtil.java 中增加如下代码：

```
public static void setSingleLine(TextView textview, boolean enabled) {
    textview.setSingleLine(enabled);
    if (enabled){
        //在单行显示时，设置支持水平滚动文本
        textview.setHorizontallyScrolling(enabled);
    }
  }
```

因为增加的属性函数名称与文本框中的属性函数名称一样，故不需要在 OdeMessages.java 中增加属性的声明和在 OdeMessages_zh_CN.properties 中添加中文字符串。

9.2 增加允许被选中复制的属性

此属性功能支持用户复制粘贴标签组件的文本到其他地方，具体实现方式如下。

在 Label.java 中增加如下代码：

```
//增加变量标识标签文本是否允许被选中复制
private boolean textIsSelectable;

@SimpleProperty(
    category = PropertyCategory.BEHAVIOR,
    description = "设置标签文本是否可被选中复制")
public boolean TextIsSelectable() {
    return textIsSelectable;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue =
"False")
@SimpleProperty
public void TextIsSelectable(boolean textIsSelectable) {
    this.textIsSelectable = textIsSelectable;
    TextViewUtil.setTextIsSelectable(view, textIsSelectable);
}
```

在 TextViewUtil.java 中增加如下代码：

```
public static void setTextIsSelectable(TextView textview, boolean enabled) {
    textview.setTextIsSelectable(enabled);
}
```

在 OdeMessages.java 中添加属性的声明：

```
@DefaultMessage("TextIsSelectable")
@Description("")
String TextIsSelectableProperties();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
TextIsSelectableProperties = 允许被选中复制
```

9.3 增加单击事件响应函数

用 TextView 响应用户单击事件，是开发应用常会用到的功能，但 App Inventor 2 提供的

标签组件不支持此功能，现增加此功能，具体实现方式如下。

在 Label.java 中增加如下代码：

```
import android.view.View.OnClickListener;

//Label 类增加对 OnClickListener 接口的继承
public final class Label extends AndroidViewComponent implements OnClickListener {
    //构造函数中注册监听单击事件函数
    public Label(ComponentContainer container) {
        super(container);
        view = new TextView(container.$context());
        …
        view.setOnClickListener(this);
    }

    //重写从 OnClickListener 接口继承的函数
    @Override
    public void onClick(View view) {
        Click();
    }

    @SimpleEvent(description = "用户单击标签事件的响应函数")
    public void Click() {
        EventDispatcher.dispatchEvent(this, "Click");
    }
}
```

因为增加的函数名称与其他组件的函数名称一样，故不需要在 OdeMessages.java 中增加函数的声明和在 OdeMessages_zh_CN.properties 中添加中文字符串。

9.4 增加跑马灯效果函数

此函数用于设置从右向左无限循环滚动标签文本，具体实现方式如下。

在 Label.java 中增加如下代码：

```
import android.text.TextUtils;

@SimpleFunction(description = "跑马灯效果")
public void Marquee() {
    TextViewUtil.setMarquee(view);
}
```

在 TextViewUtil.java 中增加如下代码：

```
public static void setMarquee(TextView textview) {
```

```
        setSingleLine(textview, true);
        setTextIsSelectable(textview, true);
        textview.setEllipsize(TextUtils.TruncateAt.MARQUEE);
        textview.setMarqueeRepeatLimit(-1);
    }
```

在 OdeMessages.java 中添加函数的声明：

```
@DefaultMessage("Marquee")
@Description("")
String MarqueeMethods();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
MarqueeMethods = 跑马灯效果
```

9.5 增加设置标签图片的函数

此函数用于设置在标签的上下左右四边显示的图片，具体实现方式如下。

在 Label.java 中增加如下代码：

```
@SimpleFunction(description = "设置在标签的上下左右四边显示的图片")
public void SetLabelDrawable(String path, int width, int height, boolean left, boolean top, boolean right,
boolean bottom) {
    Drawable drawable = null;
    Drawable drawableLeft = null;
    Drawable drawableTop = null;
    Drawable drawableRight = null;
    Drawable drawableBottom = null;

    String picturePath = (path == null) ? "" : path;

    try {
      drawable = MediaUtil.getBitmapDrawable(container.$form(), picturePath);
    } catch (IOException ioe) {
      Log.e("Label", "Unable to load " + picturePath);

      drawable = null;
    }

    if (drawable != null){
      int correctedWidth = (int)(width * container.$form().deviceDensity());
      int correctedHeight = (int)(height * container.$form().deviceDensity());
      drawable.setBounds(0, 0, correctedWidth, correctedHeight);
```

```
            if (left){
              drawableLeft = drawable;
            }

            if (top){
              drawableTop = drawable;
            }

            if (right){
              drawableRight = drawable;
            }

            if (bottom){
              drawableBottom = drawable;
            }

            view.setCompoundDrawables(drawableLeft, drawableTop, drawableRight, drawableBottom);
            view.requestLayout();
          }
        }
```

在 Drawer.js 文件的 component_method 数组中，增加如下代码，设置上述函数布尔型参数的默认值：

```
        {matchingMutatorAttributes:{component_type:"Label", method_name:"SetLabelDrawable"},
              mutatorXMLStringFunction: function(mutatorAttributes) {
                return '' +
                  '<xml>' +
                  '<block type="component_method">' +
                  //mutator generator
                  Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
                  '<value name="ARG3"><block type="logic_boolean"><title name="BOOL">TRUE</title>
                  </block></value>' +
                  '<value name="ARG4"><block type="logic_boolean"><title name="BOOL">FALSE</title>
                  </block></value>' +
                  '<value name="ARG5"><block type="logic_boolean"><title name="BOOL">FALSE</title>
                  </block></value>' +
                  '<value name="ARG6"><block type="logic_boolean"><title name="BOOL">FALSE</title>
                  </block></value>' +
                  '</block>' +
                  '</xml>';}},
```

<value …</value> 这个字段描述了对哪个参数赋值，及默认值的类型和值。

用这种方式，App Inventor 2 实现了给函数的参数赋予默认值。

在 OdeMessages.java 中添加函数的声明：

```
@DefaultMessage("SetLabelDrawable")
@Description("")
String SetLabelDrawableMethods();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
SetLabelDrawableMethods = 设置标签的图片
```

9.6 增加富文本处理函数

富文本可以对选中的部分文字单独设置字体、字号和颜色等，也可以在文本中添加图片。在 App Inventor 2 中，如果要用标签组件实现富文本的这些功能，需要把文本按 HTML 格式处理，但这样就得了解 HTML 的知识，使用起来不方便。

Android 系统提供了用于处理富文本的类 SpannableString，可以用这个类与其他类结合，实现标签组件支持富文本的功能，且不需要具备 HTML 的知识，开发人员可以很方便地使用。

在 Label.java 中增加如下代码：

```
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.runtime.util.MediaUtil;
import android.graphics.drawable.Drawable;
import android.graphics.Typeface;

import android.text.SpannableString;
import android.text.Spanned;
import android.text.TextPaint;
import android.text.method.LinkMovementMethod;
import android.text.style.BackgroundColorSpan;
import android.text.style.ClickableSpan;
import android.text.style.ForegroundColorSpan;
import android.text.style.ImageSpan;
import android.text.style.RelativeSizeSpan;
import android.text.style.StrikethroughSpan;
import android.text.style.StyleSpan;
import android.text.style.SubscriptSpan;
import android.text.style.SuperscriptSpan;
import android.text.style.URLSpan;
import android.text.style.UnderlineSpan;
import android.text.style.AbsoluteSizeSpan;

import java.io.IOException;

//增加 SpannableString 的变量
```

```
private SpannableString spannableString;

//增加图片路径的变量
private String picturePath = "";

@SimpleFunction(description = "创建富文本")
public void CreateSpannableString(String text) {
    spannableString = new SpannableString(text);
}

@SimpleFunction(description = "设置富文本的前景色")
public void SetForegroundColor(int color, int startIndex, int endIndex) {
    ForegroundColorSpan foregroundColorSpan = new ForegroundColorSpan(color);
    spannableString.setSpan(foregroundColorSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_
    EXCLUSIVE);
}

@SimpleFunction(description = "设置富文本的背景色")
public void SetBackgroundColor(int color, int startIndex, int endIndex) {
    BackgroundColorSpan backgroundColorSpan = new BackgroundColorSpan(color);
    spannableString.setSpan(backgroundColorSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_
    EXCLUSIVE);
}

@SimpleFunction(description = "设置富文本文字的字号")
public void SetAbsoluteSize(int size, int startIndex, int endIndex) {
    AbsoluteSizeSpan absoluteSizeSpan = new AbsoluteSizeSpan(size, true);
    spannableString.setSpan(absoluteSizeSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_
    EXCLUSIVE);
}

@SimpleFunction(description = "设置富文本文字的字体为粗体")
public void SetStyleBold(int startIndex, int endIndex) {
    StyleSpan styleSpan = new StyleSpan(Typeface.BOLD);
    spannableString.setSpan(styleSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_EXCLUSIVE);
}

@SimpleFunction(description = "设置富文本文字的删除线")
public void SetStrikethrough(int startIndex, int endIndex) {
    StrikethroughSpan strikethroughSpan = new StrikethroughSpan();
    spannableString.setSpan(strikethroughSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_
    EXCLUSIVE);
}

@SimpleFunction(description = "设置富文本文字的下划线")
public void SetUnderline(int startIndex, int endIndex) {
```

```
        UnderlineSpan underlineSpan = new UnderlineSpan();
        spannableString.setSpan(underlineSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_ EXCLUSIVE);
    }

    @SimpleFunction(description = "设置富文本文字的上标")
    public void SetSuperscript(int startIndex, int endIndex) {
        SuperscriptSpan superscriptSpan = new SuperscriptSpan();
        spannableString.setSpan(superscriptSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_
        EXCLUSIVE);
    }

    @SimpleFunction(description = "设置富文本文字的下标")
    public void SetSubscript(int startIndex, int endIndex) {
        SubscriptSpan subscriptSpan = new SubscriptSpan();
        spannableString.setSpan(subscriptSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_ EXCLUSIVE);
    }

    @SimpleFunction(description = "设置富文本文字的超级链接，行为参数的值为打电话、发短信、
    打开网页")
    public void SetURL(String action, String text, int startIndex, int endIndex) {
        URLSpan urlSpan;

        if (action.equals("打电话")) {
            urlSpan = new URLSpan("tel:" + text);
            spannableString.setSpan(urlSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_EXCLUSIVE);
            view.setMovementMethod(LinkMovementMethod.getInstance());
        }

        if (action.equals("发短信")) {
            urlSpan = new URLSpan("smsto:" + text);
            spannableString.setSpan(urlSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_EXCLUSIVE);
            view.setMovementMethod(LinkMovementMethod.getInstance());
        }

        if (action.equals("打开网页")) {
            urlSpan = new URLSpan(text);
            spannableString.setSpan(urlSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_EXCLUSIVE);
            view.setMovementMethod(LinkMovementMethod.getInstance());
        }
    }

    @SimpleFunction(description = "富文本中插入图片，width 和 height 是图片显示区域的宽度和高度")
    public void SetImage(String path, int width, int height, int startIndex, int endIndex) {
        picturePath = (path == null) ? "" : path;

        Drawable drawable;
```

```
        try {
            drawable = MediaUtil.getBitmapDrawable(container.$form(), picturePath);
        } catch (IOException ioe) {
            Log.e("Image", "Unable to load " + picturePath);
            drawable = null;
        }

        int correctedWidth = (int)(width * container.$form().deviceDensity());
        int correctedHeight = (int)(height * container.$form().deviceDensity());

        drawable.setBounds(0, 0, correctedWidth, correctedHeight);
        ImageSpan imageSpan = new ImageSpan(drawable);
        spannableString.setSpan(imageSpan, startIndex, endIndex, Spanned.SPAN_INCLUSIVE_EXCLUSIVE);
    }

    @SimpleFunction(description = "设置富文本为标签内容")
    public void SetSpannableString() {
        view.setText(spannableString);
        view.requestLayout();
    }
```

在 OdeMessages.java 中添加函数和参数的声明：

```
    @DefaultMessage("CreateSpannableString")
    @Description("")
    String CreateSpannableStringMethods();

    @DefaultMessage("SetForegroundColor")
    @Description("")
    String SetForegroundColorMethods();

    @DefaultMessage("SetBackgroundColor")
    @Description("")
    String SetBackgroundColorMethods();

    @DefaultMessage("SetAbsoluteSize")
    @Description("")
    String SetAbsoluteSizeMethods();

    @DefaultMessage("SetStyleBold ")
    @Description("")
    String SetStyleBoldMethods();

    @DefaultMessage("SetStrikethrough")
    @Description("")
```

```
String SetStrikethroughMethods();

@DefaultMessage("SetUnderline")
@Description("")
String SetUnderlineMethods();

@DefaultMessage("SetSuperscript")
@Description("")
String SetSuperscriptMethods();

@DefaultMessage("SetSubscript")
@Description("")
String SetSubscriptMethods();

@DefaultMessage("SetURL")
@Description("")
String SetURLMethods();

@DefaultMessage("SetImage")
@Description("")
String SetImageMethods();

@DefaultMessage("SetSpannableString")
@Description("")
String SetSpannableStringMethods();

@DefaultMessage("startIndex")
@Description("")
String startIndexParams();
@DefaultMessage("endIndex")
@Description("")
String endIndexParams();

@DefaultMessage("size")
@Description("")
String sizeParams();

@DefaultMessage("action")
@Description("")
String actionParams();
```

在 OdeMessages_zh_CN.properties 中添加函数和参数的中文字符串：

```
CreateSpannableStringMethods = 创建富文本
SetForegroundColorMethods = 设置富文本的前景色
```

```
SetBackgroundColorMethods = 设置富文本的背景色
SetAbsoluteSizeMethods = 设置富文本文字的字号
SetStyleBoldMethods = 设置富文本文字的字体为粗体
SetStrikethroughMethods = 设置富文本文字的删除线
SetUnderlineMethods = 设置富文本文字的下划线
SetSuperscriptMethods = 设置富文本文字的上标
SetSubscriptMethods = 设置富文本文字的下标
SetURLMethods = 设置富文本文字的超级链接
SetImageMethods = 富文本中插入图片
SetSpannableStringMethods = 设置富文本为标签内容

startIndexParams = 开始字符的索引值
endIndexParams = 结束字符的索引值
sizeParams = 字号
actionParams = 行为
```

9.7 实现结果

编译、运行系统后，在组件属性中，可以看到增加了“允许多行”和“允许被选中复制”两个属性，如图 9-1 所示。

工作面板中，增加了这两个属性的 getter 和 setter 函数，如图 9-2 所示。



图 9-1 新增属性

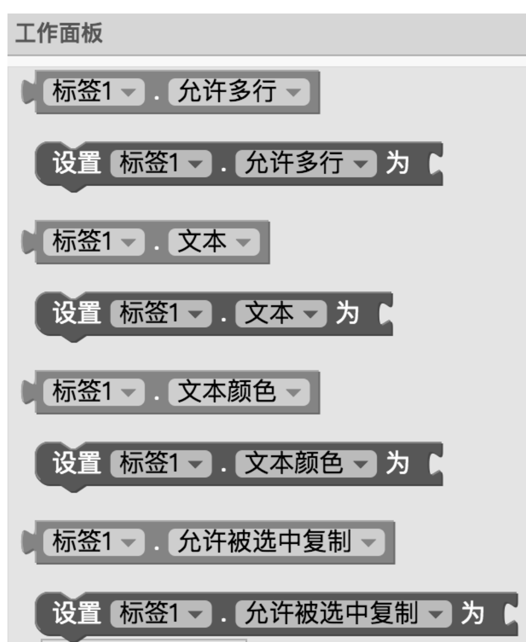


图 9-2 新增属性函数

在工作面板中，还可以看到其余新增事件响应函数和功能函数，如图 9-3～图 9-6 所示。

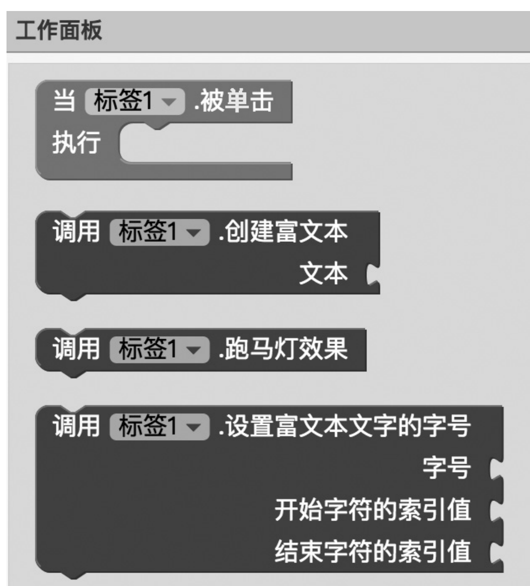


图 9-3　新增函数 1

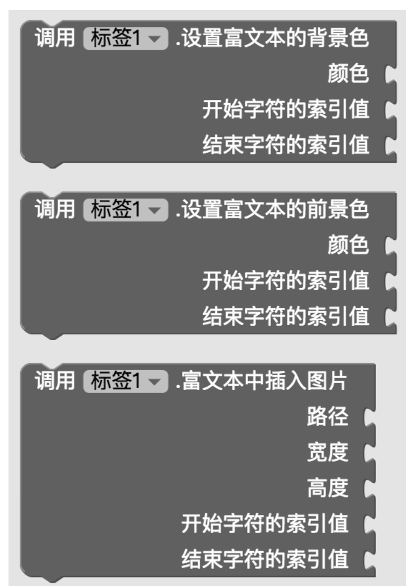


图 9-4　新增函数 2

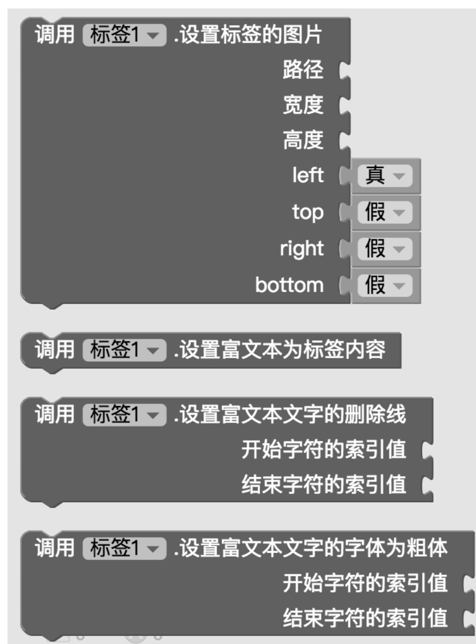


图 9-5　新增函数 3

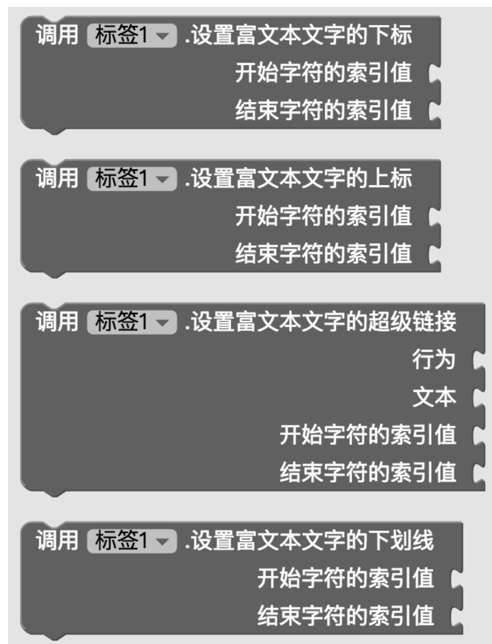


图 9-6　新增函数 4

9.8　使用示例

1）设置标签的图片、标签文本跑马灯效果和标签文本可被选中复制的代码如图 9-7 所示。

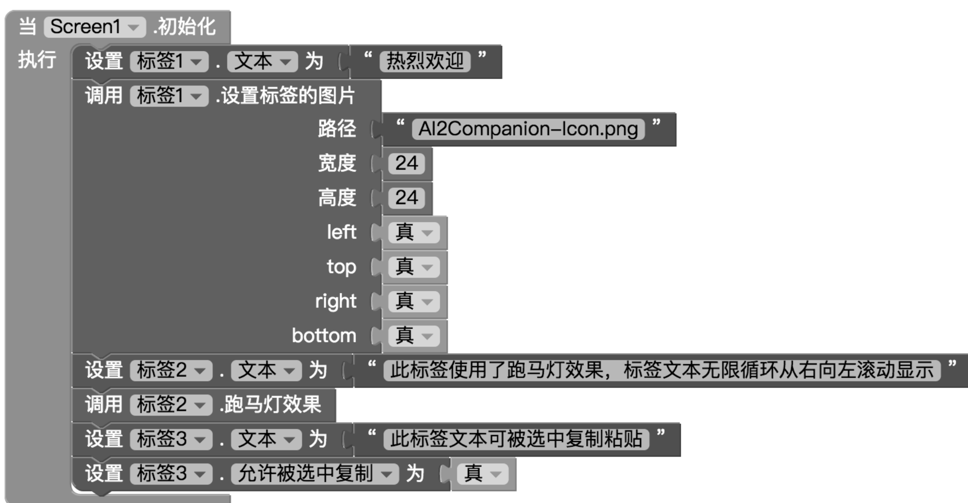


图 9-7　代码 1

实现的效果如图 9-8 和图 9-9 所示。



图 9-8　效果 1

注：图中显示的“标签 2”组件的文本正在从右向左滚动显示。

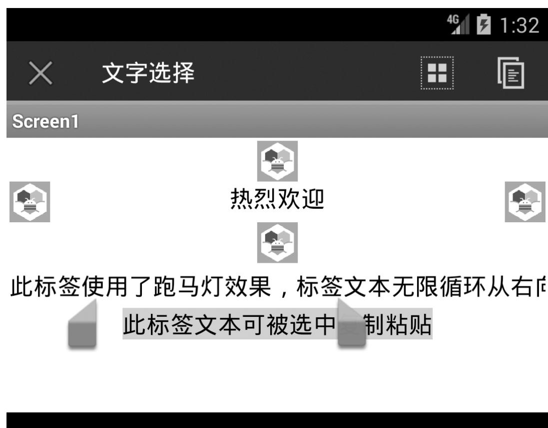


图 9-9　效果 2

注：图中显示的“标签 3”组件的文本已被选中复制。

设置标签的图片时，需要首先在工程中添加图片素材，如图 9-10 所示。

2）Android 系统提供了 RadioButton 控件（单选按钮），常用于从多个选项中选择其中之一，实现效果如图 9-11 所示。

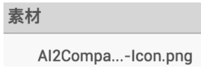


图 9-10 素材 1

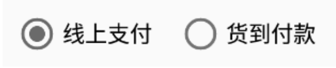


图 9-11 Android 系统的单选按钮

App Inventor 2 没有提供类似的组件，但可以利用标签组件实现同样效果，具体实现方式如下：

在工程中添加两张图片素材，如图 9-12 所示。

在界面中添加两个标签组件，按图 9-13 所示布局和设置标签文本。



图 9-12 素材 2

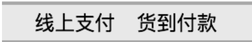


图 9-13 标签布局

具体实现代码如图 9-14～图 9-17 所示。

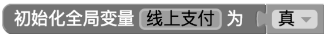


图 9-14 代码 2

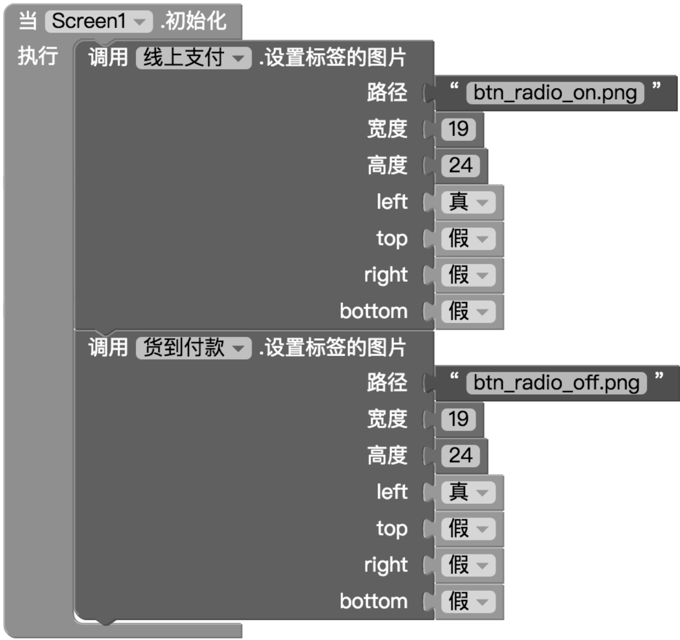
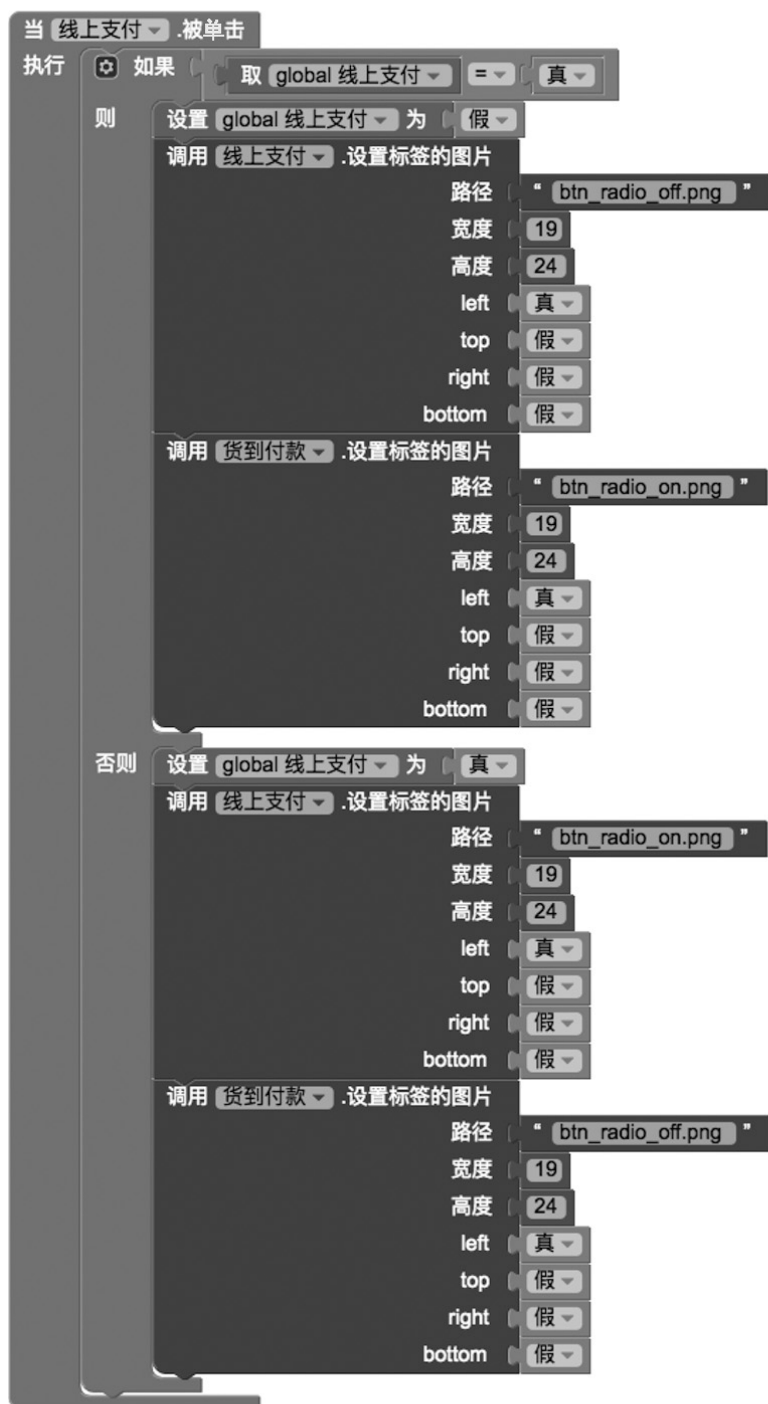


图 9-15 代码 3


当 线上支付 .被单击
执行 如果 取 global 线上支付 = 真
则 设置 global 线上支付 为 假
调用 线上支付 .设置标签的图片
路径 " btn_radio_off.png "
宽度 19
高度 24
left 真
top 假
right 假
bottom 假
调用 货到付款 .设置标签的图片
路径 " btn_radio_on.png "
宽度 19
高度 24
left 真
top 假
right 假
bottom 假
否则 设置 global 线上支付 为 真
调用 线上支付 .设置标签的图片
路径 " btn_radio_on.png "
宽度 19
高度 24
left 真
top 假
right 假
bottom 假
调用 货到付款 .设置标签的图片
路径 " btn_radio_off.png "
宽度 19
高度 24
left 真
top 假
right 假
bottom 假
图 9-16 代码 4

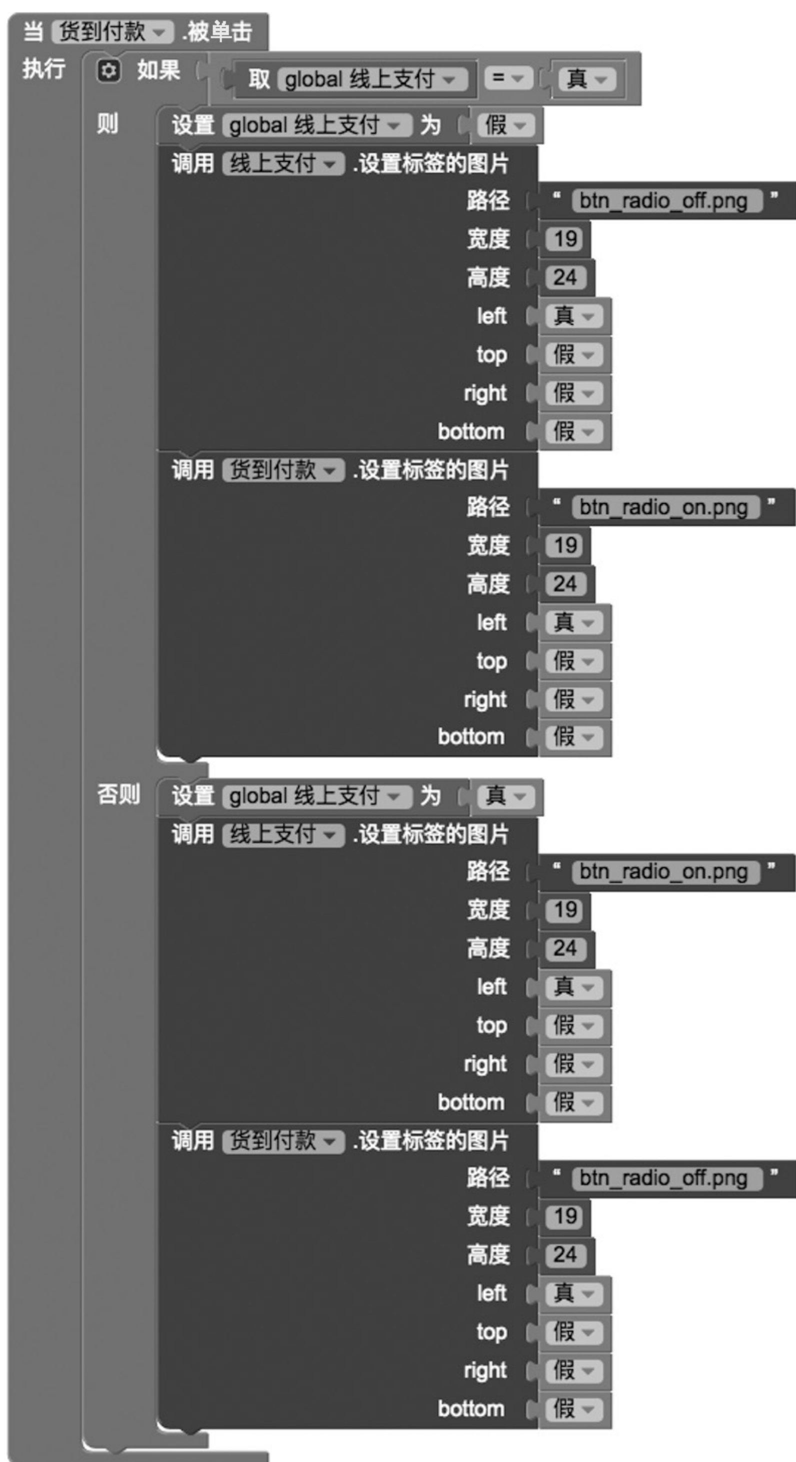


图 9-17　代码 5

实现的显示效果如图 9-18 和图 9-19 所示。

线上支付　货到付款

图 9-18　效果 3

线上支付　货到付款

图 9-19　效果 4

3）设置富文本的颜色、字号、下划线、删除线和字体的代码如图 9-20 所示。

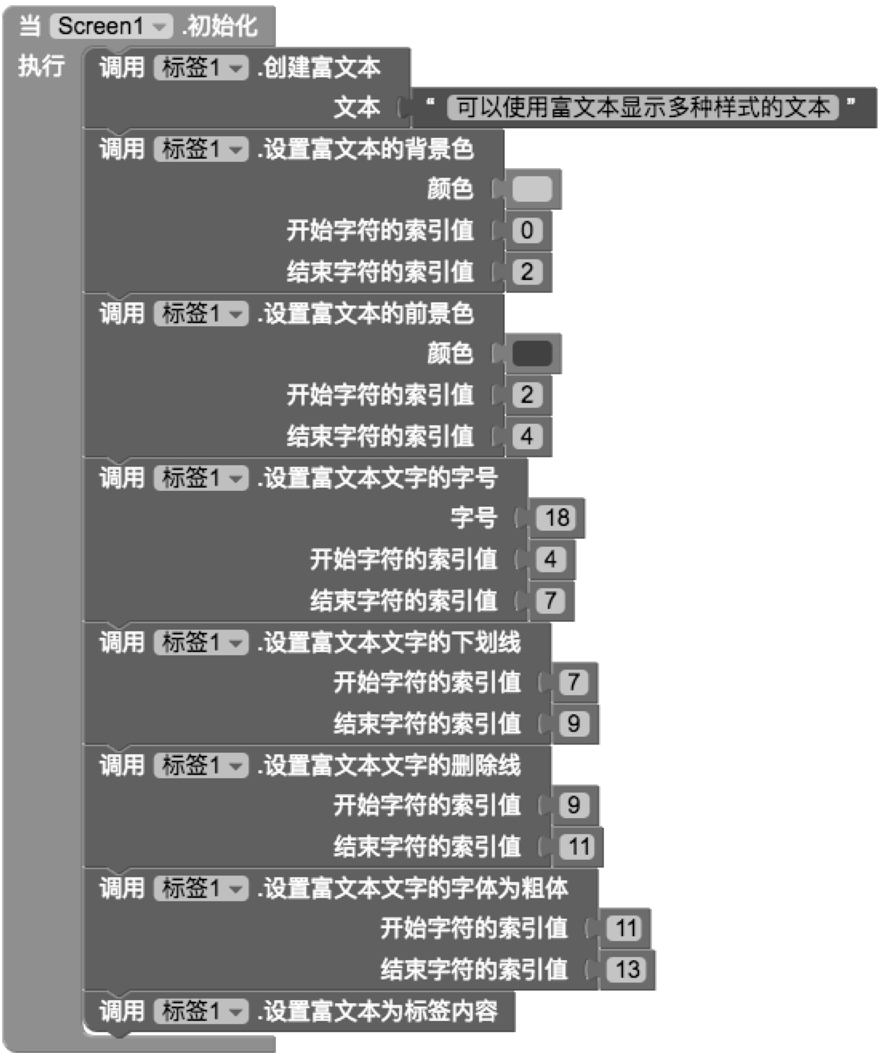


图 9-20　代码 6

实现的显示效果如图 9-21 所示。

可以使用富文本显示多种样式的文本

图 9-21　效果 5

4）设置富文本文字的上下标功能，可用于数学计算方面。示例代码如图 9-22 所示。

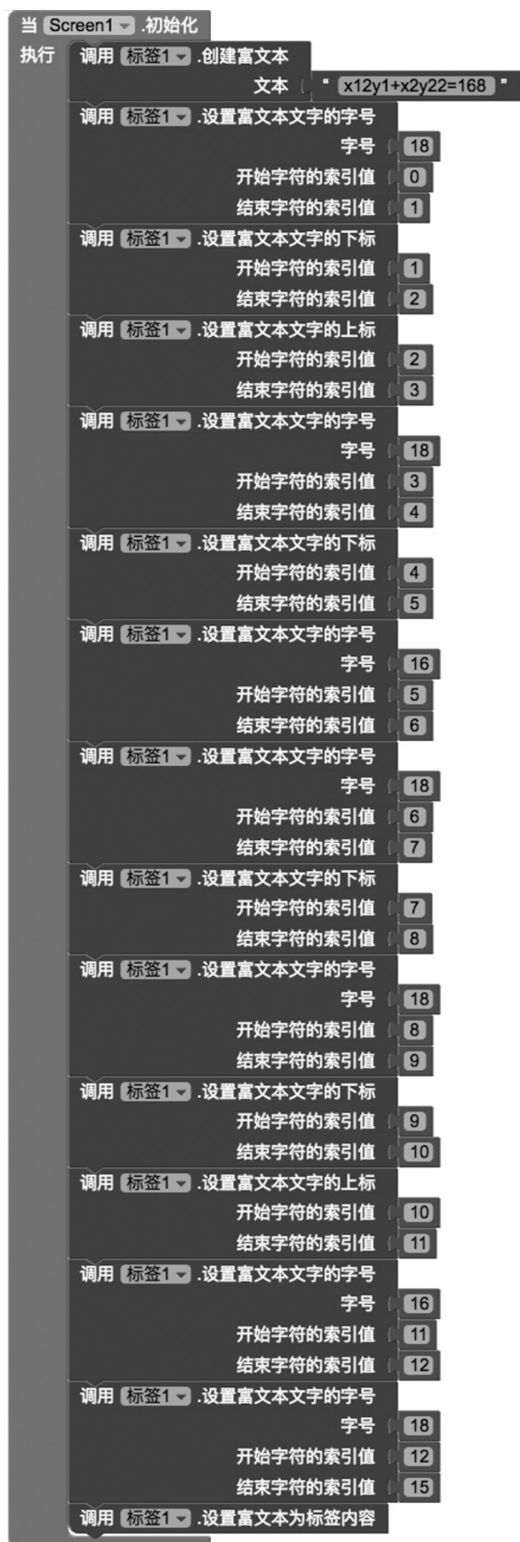

当 Screen1 .初始化
执行 调用 标签1 .创建富文本
文本 " x12y1+x2y22=168 "
调用 标签1 .设置富文本文字的字号
字号 18
开始字符的索引值 0
结束字符的索引值 1
调用 标签1 .设置富文本文字的下标
开始字符的索引值 1
结束字符的索引值 2
调用 标签1 .设置富文本文字的上标
开始字符的索引值 2
结束字符的索引值 3
调用 标签1 .设置富文本文字的字号
字号 18
开始字符的索引值 3
结束字符的索引值 4
调用 标签1 .设置富文本文字的下标
开始字符的索引值 4
结束字符的索引值 5
调用 标签1 .设置富文本文字的字号
字号 16
开始字符的索引值 5
结束字符的索引值 6
调用 标签1 .设置富文本文字的字号
字号 18
开始字符的索引值 6
结束字符的索引值 7
调用 标签1 .设置富文本文字的下标
开始字符的索引值 7
结束字符的索引值 8
调用 标签1 .设置富文本文字的字号
字号 18
开始字符的索引值 8
结束字符的索引值 9
调用 标签1 .设置富文本文字的下标
开始字符的索引值 9
结束字符的索引值 10
调用 标签1 .设置富文本文字的上标
开始字符的索引值 10
结束字符的索引值 11
调用 标签1 .设置富文本文字的字号
字号 16
开始字符的索引值 11
结束字符的索引值 12
调用 标签1 .设置富文本文字的字号
字号 18
开始字符的索引值 12
结束字符的索引值 15
调用 标签1 .设置富文本为标签内容

图 9-22　代码 7

实现的显示效果如图 9-23 所示。

5）向富文本中插入图片时，需要首先在工程中添加图片素材，如图 9-24 所示。

$$x_1{}^2y_1+x_2y_2{}^2=168$$

图 9-23　效果 6

素材

smiling1.png

smiling2.png

smiling3.png

图 9-24　素材 3

具体实现代码如图 9-25 所示。



图 9-25　代码 8

实现的显示效果如图 9-26 所示。



图 9-26　效果 7

在需要插入图片的地方，在字符串中都先用空格代替图片，然后插入图片的时候，就不会把显示的字符替代掉。

6）设置富文本文字的超级链接的代码如图 9-27 所示。

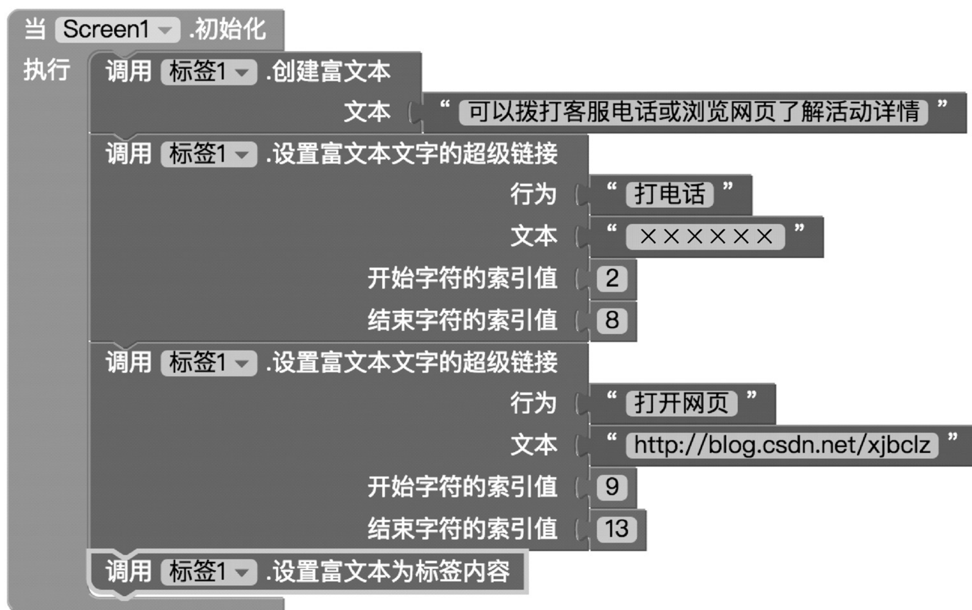


图 9-27　代码 9

实现的显示效果如图 9-28 所示。

可以拨打客服电话或浏览网页了解活动详情

图 9-28　效果 8

单击“拨打客服电话”这几个字的时候，就会调用拨号程序拨打××××××这个号码；单击“浏览网页”这几个字的时候，会调用浏览器打开 http://blog.csdn.net/xjbclz 这个网页。

第 10 章　列表显示框（Listview）的定制

10.1　修改显示字符串

App Inventor 2 提供的列表显示框，用户可以设置是否在其上显示搜索框，但搜索框中显示的提示字符串为英文，可修改为中文。

列表提示框的源码文件是 ListView.java，修改的代码如下：

```
//txtSearchBox.setHint("Search list...");
 txtSearchBox.setHint("搜索列表...");
```

单击列表选择框时，也会显示列表提示框，列表选择框的源码文件是 ListPicker.java，单击列表选择框，调用 ListPickerActivity 显示列表框，修改 ListPickerActivity.java 的代码如下：

```
//txtSearchBox.setHint("Search list...");
txtSearchBox.setHint("搜索列表...");
```

最终实现的显示效果如图 10-1 所示。



图 10-1　中文显示效果

10.2　增加显示框的显示样式

Android 系统还提供了其他形式的列表显示框，如图 10-2 所示。

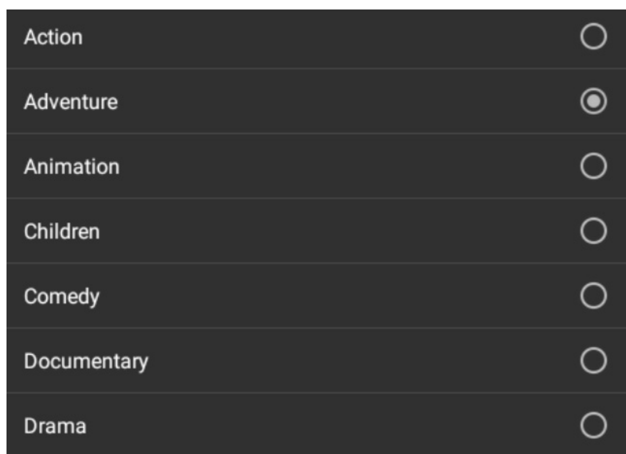


图 10-2　具有单选按钮的列表显示框 1

在每个 Item 的最右边有个单选按钮，想要实现此种形式的单选按钮，也需要修改 ListView.java 的代码：

```
//标识是否使用具有单选按钮的列表显示框，默认为 false
private boolean singleChoice = false;
```

增加设置和获取此变量值的函数：

```
@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
      defaultValue = "False")
@SimpleProperty(description = "设置是否显示具有单选按钮的列表显示框")
public void SingleChoice(boolean singleChoice) {
   this.singleChoice = singleChoice;
}

@SimpleProperty(category = PropertyCategory. APPEARANCE,
      description = "返回设置单选按钮的列表显示框的变量数值")
public boolean SingleChoice() {
   return singleChoice;
}
```

在创建列表显示框的函数中，添加设置列表显示框样式的代码：

```
public void setAdapterData(){
if (!singleChoice){
   //App Invetor 2 默认提供的列表显示框
      adapter = new ArrayAdapter<Spannable>(container.$context(), android.R.layout.simple_list_item_1,
         itemsToColoredText());
} else {

      //创建具有单选按钮的列表显示框，布局文件和上面不同
      adapter=new ArrayAdapter<Spannable>(container.$context(), android.R.layout.simple_list_item_
```

```
            single_ choice,
            itemsToColoredText());

            //Android 系统还支持在列表框中的多选，在此设置为单选模式 CHOICE_MODE_SINGLE = 1
            view.setChoiceMode(1);
        }
        view.setAdapter(adapter);

        if (!singleChoice){
            adapterCopy = new ArrayAdapter<Spannable>(container.$context(), android.R.layout.simple_
            list_item_1);
        }else{
            adapterCopy = new ArrayAdapter<Spannable>(container.$context(), android.R.layout.simple_
            list_ item_ single_choice);
        }

        for (int i = 0; i < adapter.getCount(); ++i) {
            adapterCopy.insert(adapter.getItem(i), i);
        }
    }
```

在 OdeMessages.java 中增加属性声明：

```
    @DefaultMessage("SingleChoice")
    @Description("")
    String SingleChoiceProperties();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
    SingleChoiceProperties = 显示具有单选按钮的列表显示框
```

编译、运行系统后，在列表显示框的组件属性中，可以看到增加了一个“显示具有单选按钮的列表显示框”的属性，默认为不显示，如图 10-3 所示。

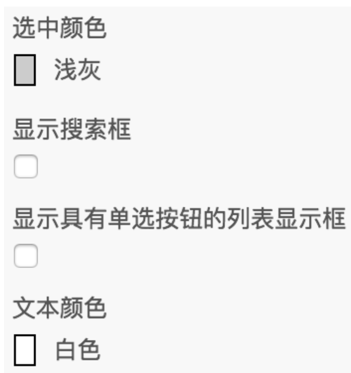


图 10-3　新增属性

在工作面板中，也可以看到增加了此属性的 getter 和 setter 函数，如图 10-4 所示。

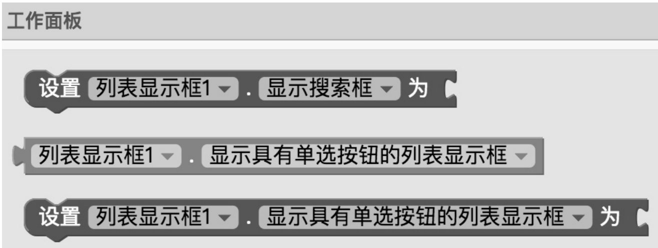


图 10-4　新增属性函数

最终实现的显示效果如图 10-5 所示。



图 10-5　具有单选按钮的列表显示框 2

第 11 章　对话框（Notifier）的定制

11.1　修改对话框的外观

系统默认创建的对话框上没有图标，且是 Android 系统比较古老的样式，很不美观，现修改源码增加图标和修改对话框的样式。

11.1.1　增加设置图标的属性

对话框的源码文件是 Notifier.java，在其中首先增加一个供用户设置是否显示图标的属性：

```
//增加标识是否显示图标的变量，默认为 true
private static boolean bIsDisplayIcon = true;
```

在创建对话框的函数中，添加设置图标的代码：

```
public static void oneButtonAlert(Activity activity,String message, String title, String buttonText) {
    …
    if (bIsDisplayIcon){
        alertDialog.setIcon(android.R.drawable.sym_def_app_icon);
    }


    …
}

public static void twoButtonDialog(Activity activity, String message, String title,
        final string button1 Text, final string button 2 Text, boolean cancelable,
        final Runnable positiveAction, final Runnable negativeAction, final Runnable cance(Action){
    …
    if (bIsDisplayIcon){
        alertDialog.setIcon(android.R.drawable.sym_def_app_icon);
    }


    …
}

private void textInputDialog(String message, String title, boolean cancelable) {
    …
```

```
            if (bIsDisplayIcon){
                alertDialog.setIcon(android.R.drawable.sym_def_app_icon);
            }

            …
        }
```

增加设置和获取属性值的函数：

```
        @SimpleProperty(description = "设置对话框是否显示图标",
                category = PropertyCategory.APPEARANCE)
        public boolean IsDisplayIcon() {
            return bIsDisplayIcon;
        }


        @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
                defaultValue = "True")
        @SimpleProperty
        public void IsDisplayIcon(boolean bIsDisplayIcon) {
            bIsDisplayIcon = bIsDisplayIcon;
        }
```

在 OdeMessages.java 中增加属性声明：

```
        @DefaultMessage("IsDisplayIcon")
        @Description("")
        String IsDisplayIconProperties();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
        IsDisplayIconProperties = 设置对话框是否显示图标
```

编译、运行系统后，在对话框的组件属性中，可以看到增加了一个“设置对话框是否显示图标”的属性，默认为显示，如图 11-1 所示。



图 11-1　新增属性 1

在工作面板中，也可以看到增加了此属性的 getter 和 setter 函数，如图 11-2 所示。

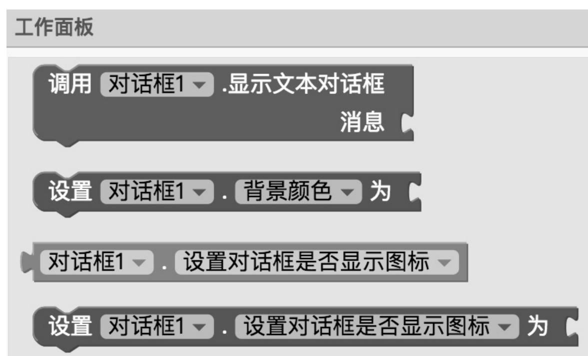


图 11-2　新增属性函数 1

对话框的显示效果如图 11-3～图 11-5 所示。



图 11-3　显示效果 1



图 11-4　显示效果 2



图 11-5　显示效果 3

11.1.2　增加修改对话框显示风格的属性

Google 推荐 Android App 的显示风格是 Material Design 风格，现增加一个可设置对话框显示风格为 Material Design 风格的属性。

```
//增加标识是否使用 Material Design 风格的变量，默认为 false
private static boolean bIsMaterialDesign = false;
```

在创建对话框的函数中，添加设置显示风格的代码：

```
    public static void oneButtonAlert(Activity activity,String message, String title, String buttonText) {
        AlertDialog alertDialog = null;

        if (bIsMaterialDesign){
            alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
            Alert). create();
        } else {
            alertDialog = new AlertDialog.Builder(activity).create();
        }

        …
    }

    public static void twoButtonDialog(Activity activity, String message, String title,
            final string button1 Text, final string button2 Text, boolean cancelable,
            final Runnable positiveAction, final Runnable negativeAction, final Runnable cance(Action){
        AlertDialog alertDialog = null;

        if (bIsMaterialDesign){
            alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
            Alert).create();
        } else {
            alertDialog = new AlertDialog.Builder(activity).create();
        }

        …
    }

    private void textInputDialog(String message, String title, boolean cancelable) {
        AlertDialog alertDialog = null;

        if (bIsMaterialDesign){
            alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
            Alert).create();
        } else {
            alertDialog = new AlertDialog.Builder(activity).create();
        }

        …
    }
```

增加设置和获取属性值的函数：

```
    @SimpleProperty(description = "设置是否创建 Material Design 风格的对话框",
            category = PropertyCategory.APPEARANCE)
    public boolean IsMaterialDesign() {
        return bIsMaterialDesign;
```

```
    }

    @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
            defaultValue = "False")
    @SimpleProperty
    public void IsMaterialDesign(boolean bIsMaterialDesign) {
        this.bIsMaterialDesign = bIsMaterialDesign;
    }
```

在 OdeMessages.java 中增加属性声明：

```
    @DefaultMessage("IsMaterialDesign")
    @Description("")
    String IsMaterialDesignProperties();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
    IsMaterialDesignProperties = 设置是否创建 Material Design 风格的对话框
```

编译、运行系统后，在对话框的组件属性中，可以看到增加了一个“设置是否创建 Material Design 风格的对话框”的属性，默认为不创建，如图 11-6 所示。



图 11-6　新增属性 2

在工作面板中，也可以看到增加了此属性的 getter 和 setter 函数，如图 11-7 所示。



图 11-7　新增属性函数 2

在设置创建 Material Design 风格的对话框后，对话框的显示效果如图 11-8～图 11-10 所示。



图 11-8　显示效果 4



图 11-9　显示效果 5



图 11-10　显示效果 6

11.2　增加显示函数

对话框组件默认已经提供了几个函数供用户使用，如图 11-11～图 11-13 所示。

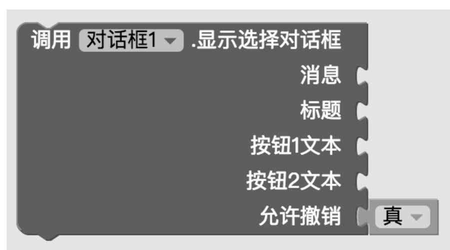


图 11-11　原有“显示选择对话框”函数



图 11-12　原有“显示消息对话框”函数

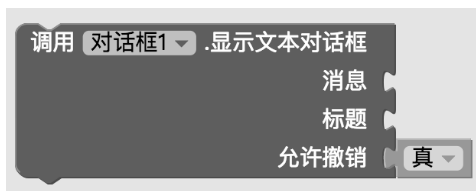


图 11-13　原有“显示文本对话框”函数

这几个函数都需要开发人员输入多个参数，使用不方便。

通常应用中在各个页面使用的提示框、标题和按钮文字都是固定的，只是消息内容有变化，故可增加几个显示对话框的函数，实现和上述函数同样的功能，但只需用户输入消息内容，不需要输入标题和按钮文本，减少开发人员的输入，提高开发效率。

在 Notifier.java 中增加以下代码：

```
@SimpleFunction(description = "显示提示框，标题文字-提示，按钮文字-确定")
public void ShowMessageDialogNew(String message) {
  oneButtonAlertNew(activity, message);
}

// This method is declared static, with an explicit activity input, so that other
// components can use it
public static void oneButtonAlertNew(Activity activity,String message) {
  Log.i(LOG_TAG, "One button alert " + message);
  AlertDialog alertDialog = null;

  if (bIsMaterialDesign){
    alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
    Alert).create();
  } else {
    alertDialog = new AlertDialog.Builder(activity).create();
  }

  if (bIsDisplayIcon){
    alertDialog.setIcon(android.R.drawable.ic_dialog_info);
  }

  alertDialog.setTitle("提示");
  // prevents the user from escaping the dialog by hitting the Back button
  alertDialog.setCancelable(false);
  alertDialog.setMessage(stringToHTML(message));

  alertDialog.setButton(DialogInterface.BUTTON_NEUTRAL,
    "确定", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
    }});
  alertDialog.show();
}

@SimpleFunction(description = "显示选择框，标题文字-提示，按钮文字-确定和取消")
public void ShowChooseDialogNew(String message) {
  twoButtonDialogNew(activity,
      message,
      "提示",
```

```
            "确定",
            new Runnable() {public void run() {AfterChoosing("确定");}},
            new Runnable() {public void run() {AfterChoosing(activity.getString(android.R.string.cancel));}}
            );
    }

    // This method takes three runnables that specify the actions to be performed
    // when the buttons are pressed. It's declared static with an explicit activity input
    // so that other components can use it.
    public static void twoButtonDialogNew(Activity activity, String message,    String title,
          final String button1Text, final Runnable positiveAction, final Runnable cancelAction) {
      Log.i(LOG_TAG, "ShowChooseDialog: " + message);
      AlertDialog alertDialog = null;

      if (bIsMaterialDesign){
        alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
        Alert).create();
      } else {
        alertDialog = new AlertDialog.Builder(activity).create();
      }

      if (bIsDisplayIcon){
        alertDialog.setIcon(android.R.drawable.ic_dialog_info);
      }

      alertDialog.setTitle(title);
      // prevents the user from escaping the dialog by hitting the Back button
      alertDialog.setCancelable(false);
      alertDialog.setMessage(stringToHTML(message));


      alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, button1Text,
            new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
          positiveAction.run();
        }
      });

      // TODO(hal): It would be safer and more consistent to pass in cancelButtonText as a parameter.
      final String cancelButtonText = activity.getString(android.R.string.cancel);
      alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, cancelButtonText,
        new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
          cancelAction.run();
        }
      });
```

```
    alertDialog.show();
}

@SimpleFunction(description = "显示文本输入框，标题文字-提示，按钮文字-确定和取消")
public void ShowTextDialogNew(String message) {
    textInputDialogNew(message);
}
// TODO(hal):    It would be cleaner to define this in terms of oneButtonAlert and generalize
// oneButtonAlert so it can be used both for messages and text input.    We could have merged
// this method into ShowTextDialog, but that would make it harder to do the generalization.
private void textInputDialogNew(String message) {
    AlertDialog alertDialog = null;

    if (bIsMaterialDesign){
        alertDialog = new AlertDialog.Builder(activity, android.R.style.Theme_Material_Light_Dialog_
        Alert).create();
    } else {
        alertDialog = new AlertDialog.Builder(activity).create();
    }

    if (bIsDisplayIcon){
        alertDialog.setIcon(android.R.drawable.ic_input_add);
    }

    alertDialog.setTitle("提示");
    alertDialog.setMessage(stringToHTML(message));
    // Set an EditText view to get user input
    final EditText input = new EditText(activity);
    alertDialog.setView(input);
    // prevents the user from escaping the dialog by hitting the Back button
    alertDialog.setCancelable(false);
    alertDialog.setButton(DialogInterface.BUTTON_POSITIVE, "确定",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    HideKeyboard((View) input);
                    AfterTextInput(input.getText().toString());
                }
            });


    final String cancelButtonText = activity.getString(android.R.string.cancel);
    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, cancelButtonText,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                HideKeyboard((View) input);
```

```
                //User pressed CANCEL. Raise AfterTextInput with CANCEL
                AfterTextInput(cancelButtonText);
            }
        });
    alertDialog.show();
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("ShowChooseDialogNew")
@Description("")
String ShowChooseDialogNewMethods();

@DefaultMessage("ShowMessageDialogNew")
@Description("")
String ShowMessageDialogNewMethods();

@DefaultMessage("ShowTextDialogNew")
@Description("")
String ShowTextDialogNewMethods();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
ShowChooseDialogNewMethods = 显示选择对话框
ShowMessageDialogNewMethods = 显示消息对话框
ShowTextDialogNewMethods = 显示文本对话框
```

编译、运行系统后，在工作面板中，可以看到增加了 3 个新函数，如图 11-14～图 11-16 所示。

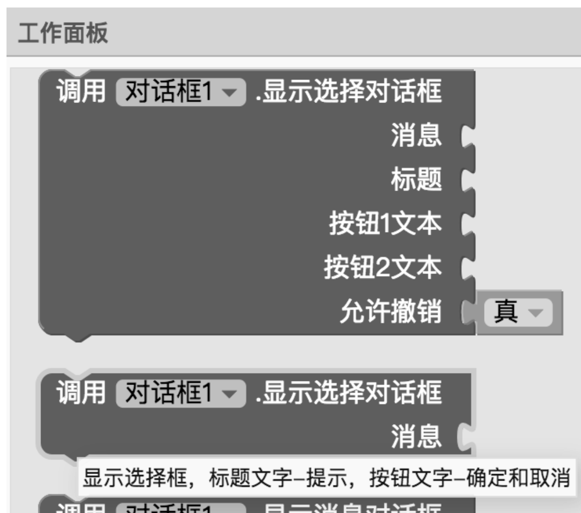


图 11-14　新增“显示选择对话框”函数

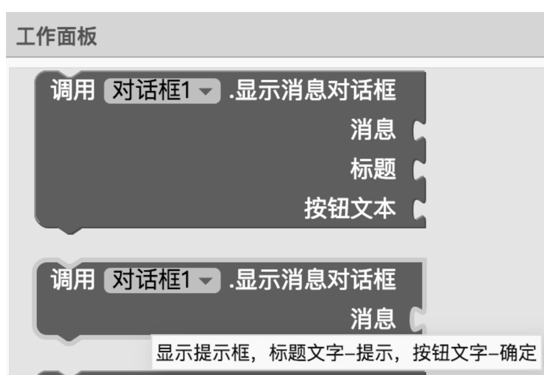


图 11–15　新增“显示消息对话框”函数

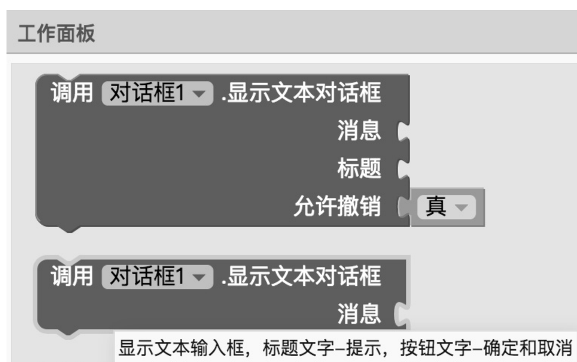


图 11–16　新增“显示文本对话框”函数

11.3　修改日志功能

11.3.1　日志功能介绍

对话框组件提供了 3 个打印日志信息函数，如图 11–17 所示。

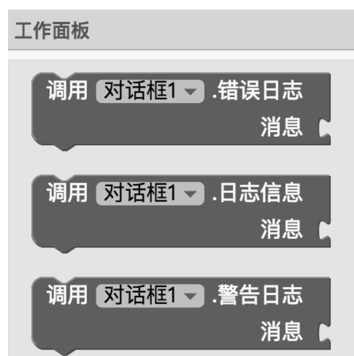


图 11–17　打印日志信息函数

对应的源码如下：

```
@SimpleFunction(description = "Writes an error message to the Android system log. " +
    "See the Google Android documentation for how to access the log.")
public void LogError(String message) {
    Log.e(LOG_TAG, message);
}

@SimpleFunction(description = "Writes a warning message to the Android log. " +
    "See the Google Android documentation for how to access the log.")
public void LogWarning(String message) {
    Log.w(LOG_TAG , message);
}

@SimpleFunction(description = "Writes an information message to the Android log.")
public void LogInfo(String message) {
    Log.i(LOG_TAG, message);
}
```

从源码可看出，这 3 个函数是利用了 Android 系统自身的 Log 机制，打印日志。

在应用中增加打印日志的功能，代码如图 11-18 所示。

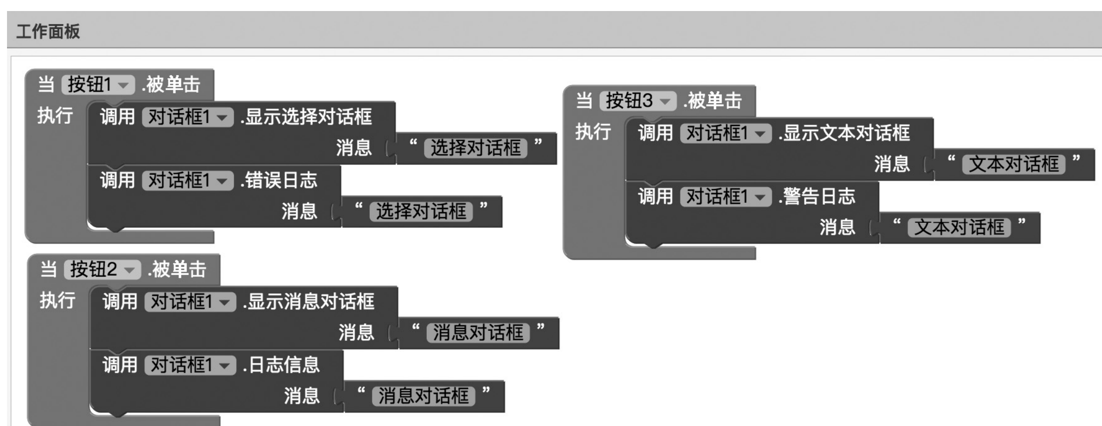


图 11-18　代码

如果要看输出的日志，在电脑端启动 Android 开发工具，如 Android Studio，然后利用 ADB 连接手机或模拟器，运行应用，就可以看到打印的日志信息了，如图 11-19 所示。



图 11-19　日志信息

Log 打印函数中的 TAG，用于筛选想要查看的日志，在 Notifier.java 中定义的 TAG 如下：

```
private static final String LOG_TAG = "Notifier";
```

可以看到输出了 5 条日志，其中两条是 Notifier.java 函数中的日志打印代码输出的：

```
public static void oneButtonAlertNew(Activity activity,String message) {
    Log.i(LOG_TAG, "One button alert " + message);
}

public static void twoButtonDialogNew(Activity activity, String message,    String title,
        final String button1Text,    final Runnable positiveAction, final Runnable cancelAction) {
    Log.i(LOG_TAG, "ShowChooseDialog: " + message);
}
```

11.3.2 增加设置日志开关属性

打印日志功能主要是在应用的开发阶段使用的，发布给用户使用的版本中通常是关闭打印日志功能的，有可能在程序中多处都添加了打印日志的函数，如果再一个个删除，比较麻烦，可以添加一个属性控制是否打印日志。

```
//增加标识是否打印日志的变量，默认为 true
private boolean bIsPrintLog = true;
```

增加设置和获取此变量值的函数：

```
@SimpleProperty(description = "设置是否打印日志",
        category = PropertyCategory.BEHAVIOR)
public boolean IsPrintLog() {
    return bIsPrintLog;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
        defaultValue = "True")
@SimpleProperty
public void IsPrintLog(boolean bIsPrintLog) {
    this.bIsPrintLog = bIsPrintLog;
}
```

在函数中用此变量控制是否打印日志：

```
@SimpleFunction(description = "Writes an error message to the Android system log. " +
      "See the Google Android documentation for how to access the log.")
public void LogError(String message) {
    if (bIsPrintLog){
        Log.e(LOG_TAG, message);
```

```
        }
    }


    @SimpleFunction(description = "Writes a warning message to the Android log. " +
          "See the Google Android documentation for how to access the log.")
    public void LogWarning(String message) {
        if (bIsPrintLog){
          Log.w(LOG_TAG, message);
        }
    }

    @SimpleFunction(description = "Writes an information message to the Android log.")
    public void LogInfo(String message) {
        if (bIsPrintLog){
          Log.i(LOG_TAG , message);
        }
    }
```

在 OdeMessages.java 中增加属性声明：

```
    @DefaultMessage("IsPrintLog")
    @Description("")
    String IsPrintLogProperties();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
    IsPrintLogProperties = 是否打印日志
```

编译、运行系统后，在对话框的组件属性中，可以看到增加了一个“是否打印日志”的属性，默认为打印，如图 11-20 所示。



图 11-20　新增属性 3

在工作面板中，也可以看到增加了此属性的 getter 和 setter 函数，如图 11-21 所示。

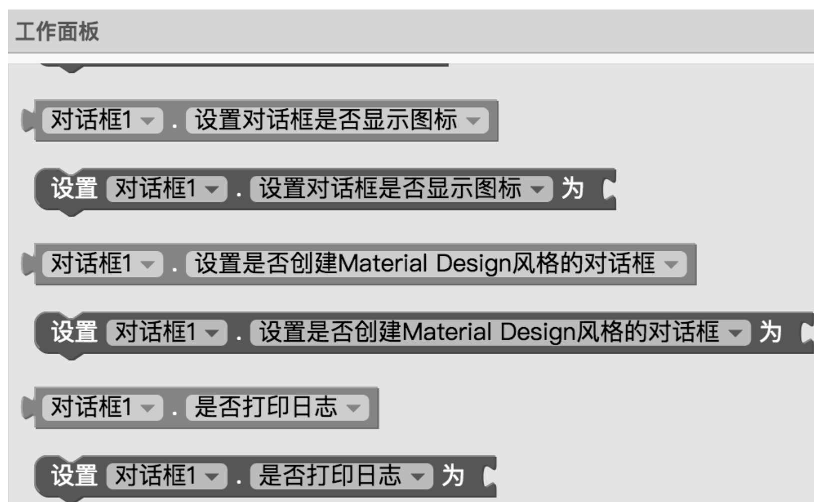


图 11-21　新增属性函数 3

第 12 章　输入框的定制

12.1　输入框的共性定制

输入框也是常使用的组件，App Inventor 2 默认提供了两种输入框：密码输入框（PasswordTextBox）和文本输入框（TextBox），这两个组件已经具有了一些属性和功能函数，但有些常用的属性和功能函数并没有，如对于用户输入字符个数的限制和清空输入框等功能。

密码输入框和文本输入框在源码中对应的类分别是 PasswordTextBox 和 TextBox，两者有共同的基类 TextBoxBase，只要在 TextBase.java 中增加上述属性和功能，密码输入框和文本输入框会自动继承并拥有新增的属性和功能。具体修改如下。

12.1.1　增加设置字符串长度的属性和核查函数

在 TextBase.java 中的增加如下代码：

```
//增加表示字符串长度的变量
private int textLength;

@SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "Set the length of text")
public int TextLength() {
   return textLength;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_ INTEGER,
 defaultValue = "0")
@SimpleProperty
public void TextLength(int length) {
   this.textLength = length;
}

@SimpleFunction(
      description = "核查文本框中的字符串长度是否和设置的字符串长度一致")
public boolean CheckTextLength() {
        if (textLength != 0){
               if (Text().length() == textLength){
```

```
                return true;
            }else{
                return false;
            }
        }else{
            //如果属性的值为 0，则表示长度没有限制
            return true;
        }
    }
```

在 OdeMessages.java 中添加属性和函数的声明：

```
@DefaultMessage("TextLength")
@Description("")
String TextLengthProperties();

@DefaultMessage("CheckTextLength")
@Description("")
String CheckTextLengthMethods();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
TextLengthProperties = 文本长度
CheckTextLengthMethods = 核查文本长度
```

编译、运行系统后，可以看到在密码输入框和文本输入框的组件属性中，都多了一个“文本长度”的属性，默认值为 0，如图 12-1 所示。

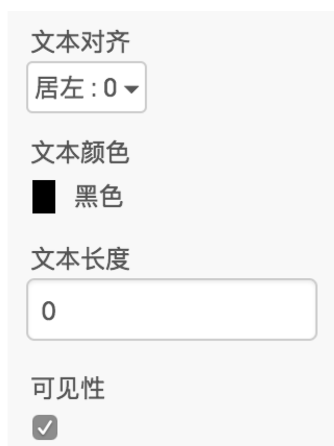


图 12-1　新增属性 1

在工作面板中，也可以看到增加了属性的 getter 和 setter 函数，如图 12-2 和图 12-3 所示。



图 12-2　新增属性函数 1



图 12-3　新增属性函数 2

在两者的工作面板中，同样可以看到都多了个“核查文本长度”的函数，如图 12-4 和图 12-5 所示。



图 12-4　新增功能函数 1

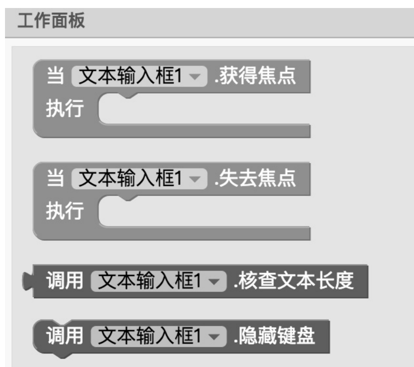


图 12-5　新增功能函数 2

使用示例代码如图 12-6 所示。



图 12-6　代码

12.1.2　增加清空输入框的函数

在 TextBase.java 中增加如下代码：

```
@SimpleFunction(description = "删除文本输入框中的所有字符")
public void DeleteAllText() {
    TextViewUtil.setText(view, "");
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("DeleteAllText")
@Description("")
String DeleteAllTextMethods();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
DeleteAllTextMethods = 清空输入框
```

编译、运行系统后，在密码输入框和文本输入框的工作面板中，都可以看到增加了一个“清空输入框”的函数，如图 12-7 和图 12-8 所示。



图 12-7　新增功能函数 3



图 12-8　新增功能函数 4

12.2　密码输入框（PasswordTextBox）的定制

用户在输入密码的时候，常想看下具体输入的内容，可以增加设置密码明文或密文显示的函数，方便用户使用。

密码输入框的源码文件是 PasswordTextBox.java，在其中增加如下代码：

```
import com.google.appinventor.components.annotations.SimpleFunction;
import android.text.method.HideReturnsTransformationMethod;

@SimpleFunction(description = "设置密码明文显示")
public void ShowPassword() {
    view.setTransformationMethod(HideReturnsTransformationMethod.getInstance());
}

@SimpleFunction(description = "设置密码密文显示")
public void HidePassword() {
    view.setTransformationMethod(PasswordTransformationMethod.getInstance());
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("ShowPassword")
@Description("")
String ShowPasswordMethods();

@DefaultMessage("HidePassword")
@Description("")
String HidePasswordMethods();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
ShowPasswordMethods = 显示密码
HidePasswordMethods = 隐藏密码
```

编译、运行系统后，在密码输入框的工作面板中，可以看到增加了“隐藏密码”和“显示密码”两个函数，如图 12-9 所示。



图 12-9　新增功能函数 5

12.3 文本输入框（TextBox）的定制

许多应用都需要用户输入电子邮箱地址，现增加一个属性，开发人员可设置文本输入框只用于输入电子邮箱地址，在 TextBox.java 中增加如下代码：

```
//属性变量
private boolean acceptsEmailAddressOnly;

@SimpleProperty(
        category = PropertyCategory.BEHAVIOR,
        description = "If true, then this text box accepts only emailaddress as keyboard input.")
public boolean EmailAddressOnly() {
    return acceptsEmailAddressOnly;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue = "False")
@SimpleProperty(
        description = "If true, then this text box accepts only emailaddress as keyboard input.")
public void EmailAddressOnly(boolean acceptsEmailAddressOnly) {
    if (acceptsEmailAddressOnly) {
        acceptsNumbersOnly = false;
        view.setInputType(
            InputType.TYPE_CLASS_TEXT |
            InputType.TYPE_TEXT_VARIATION_EMAIL_ADDRESS);
    } else {
        view.setInputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_FLAG_
        MULTI_ LINE);
    }
    this.acceptsEmailAddressOnly = acceptsEmailAddressOnly;
}
```

在 OdeMessages.java 中添加属性的声明：

```
@DefaultMessage("EmailAddressOnly")
@Description("")
String EmailAddressOnlyProperties();
```

在 OdeMessages_zh_CN.properties 中添加中文字符串：

```
EmailAddressOnlyProperties = 仅限电子邮箱
```

编译、运行系统后，可以看到在组件属性中，多了一个“仅限电子邮箱”的属性，默认值为不选中，如图 12-10 所示。



图 12-10　新增属性 2

在工作面板中也可以看到增加了属性的 getter 和 setter 函数，如图 12-11 所示。



图 12-11　新增属性函数 3

第 13 章　Web 浏览框（WebView）的定制

13.1　增加拨号功能

目前许多网页上都显示有电话号码，供用户在浏览网页的时候，可以直接拨打电话，但 App Inventor 2 提供的 Web 浏览框不支持此功能，使用不方便，现修改源码使其支持此功能。

Web 浏览框的源码文件是 WebViewer.java，首先需要在此文件中添加申请拨打电话权限的代码：

```
@UsesPermissions(permissionNames = "android.permission.INTERNET, android.permission.CALL_PHONE")
```

创建浏览框时，会执行如下代码：

```
private void resetWebViewClient() {
    if (SdkLevel.getLevel() >= SdkLevel.LEVEL_FROYO) {
      webview.setWebViewClient(FroyoUtil.getWebViewClient(ignoreSslErrors, followLinks, container.
      $form(), webview.getContext(), this));
    } else {
      webview.setWebViewClient(new WebViewerClient());
    }
}
```

目前用户使用的 Android 设备的系统版本都大于 SdkLevel.LEVEL_FROYO，所以在创建 Web 浏览框时，还会调用 FroyoUtil.java 里的 getWebViewClient 函数：

```
public static WebViewClient getWebViewClient(final boolean ignoreErrors,
    final boolean followLinks, final Form form, final Context context, final Component component) {
    return new WebViewClient() {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
       //判断 url 如果以 tel：开始，则意味此 url 中包含了电话号码，单击此 url，调用系统拨号功能
        if (url.startsWith("tel:")) {
            Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
            context.startActivity(intent);

          return true;
        }
```

```
                return !followLinks;
            }

            @Override
            public void onReceivedSslError(WebView view, SslErrorHandler handler, SslError error) {
                if (ignoreErrors) {
                    handler.proceed();
                } else {
                    handler.cancel();
                    form.dispatchErrorOccurredEvent(component, "WebView",
                        ErrorMessages.ERROR_WEBVIEW_SSL_ERROR);
                }
            }
        };
    }
```

如图 13-1 所示的网页上有技术支持热线这个电话号码，右边有个拨号图标，单击此图标，系统就会调用设备上的拨号应用，显示如图 13-2 所示的拨号界面。



图 13-1　网页

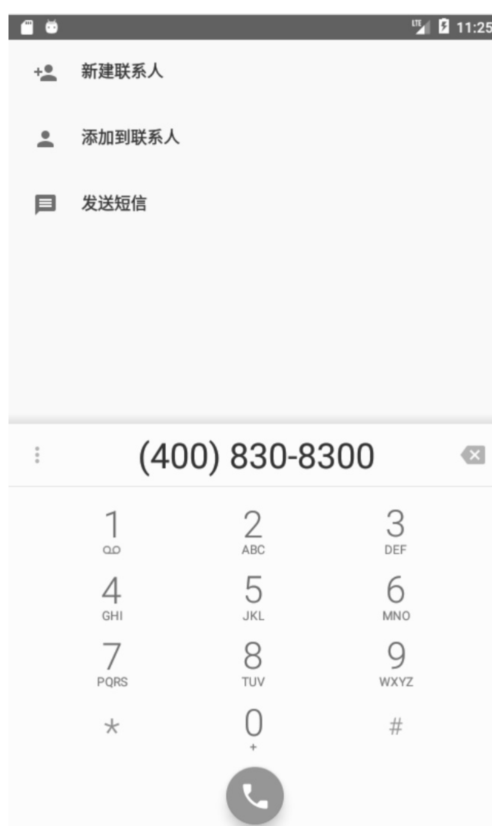


图 13-2　拨号界面

除了增加拨号功能外，也可以通过判断 url 是否以 sms:开头，增加发送短信的功能。

13.2 返回键功能的处理

用 App Inventor 2 开发的 App，组件都是附着在 Screen 上，用户单击按键，也是 Screen 先接收按键事件，对于返回键的处理代码如下（代码在 Form.java 中）：

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
   if (keyCode == KeyEvent.KEYCODE_BACK) {
      if (!BackPressed()) {
         boolean handled = super.onKeyDown(keyCode, event);
         AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
         return handled;
      } else {
         return true;
      }
   }
   return super.onKeyDown(keyCode, event);
}

@Override
protected void onStop() {
   super.onStop();
   Log.i(LOG_TAG, "Form " + formName + " got onStop");
   for (OnStopListener onStopListener : onStopListeners) {
      onStopListener.onStop();
   }
}

@Override
protected void onDestroy() {
   super.onDestroy();
   // for debugging and future growth
   Log.i(LOG_TAG, "Form " + formName + " got onDestroy");

   // Unregister events for components in this form.
   EventDispatcher.removeDispatchDelegate(this);

   for (OnDestroyListener onDestroyListener : onDestroyListeners) {
      onDestroyListener.onDestroy();
   }
}
```

当单击返回键的时候，会关闭当前应用的 Screen，退出应用，返回到系统桌面。

在用 Web 浏览框浏览网页的时候，这样处理的用户体验非常糟糕，用户在查看完第三级网页的时候，想再看下第二级网页，结果一单击返回键，直接就退出应用了。

可以在 Form.java 中，对 Web 浏览框的情况做特殊处理，代码如下：

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
   if (keyCode == KeyEvent.KEYCODE_BACK) {

      //判断是否可以返回上一级网页，如果可以则调用 Web 浏览框的 goBack 函数；如果不能再
      //返回，则执行原有代码
      if (WebViewer.isCanGoBack()) {
         WebViewer.goBack();

         return true;
      }

      if (!BackPressed()) {
         boolean handled = super.onKeyDown(keyCode, event);
         AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
         return handled;
      } else {
         return true;
      }
   }
   return super.onKeyDown(keyCode, event);
}
```

在 WebViewer.java 中增加两个静态函数，供 Form.java 调用：

```
public static boolean isCanGoBack() {
   if (webview == null){
      return false;
   } else {
      return webview.canGoBack();
   }
}

public static void goBack() {
      webview.goBack();
}
```

13.3 增加加载网页事件响应函数

在使用应用加载网页的时候，由于网络等原因，可能需要等待一会才能全部加载完网页；最好是在开始加载网页的时候，给用户提示信息，加载完成后，提示信息自动消失，以改善用户体验。

在 FroyoUtil.java 中增加如下代码：

```
import android.graphics.Bitmap;

public static WebViewClient getWebViewClient(final boolean ignoreErrors,
      final boolean followLinks, final Form form, final Context context, final Component component) {
         return new WebViewClient() {
         …
         //开始加载网页时会调用的函数
         @Override
         public void onPageStarted(WebView view, String url, Bitmap favicon) {
               super.onPageStarted(view, url, favicon);

               form.onPageStarted(url);
         }

         //完成网页加载时调用的函数
         @Override
         public void onPageFinished(WebView view, String url) {
               super.onPageFinished(view, url);

               form.onPageFinished(url);
         }
               …
      };
}
```

在 Form.java 中增加如下代码：

```
@SimpleEvent(description = "开始加载网页的响应函数")
public void OnPageStarted(String url) {
      EventDispatcher.dispatchEvent(this, "OnPageStarted", url);
}

public void onPageFinished(String url) {
      OnPageFinished(url);
}

@SimpleEvent(description = "完成网页加载的响应函数")
public void OnPageFinished(String url) {
      EventDispatcher.dispatchEvent(this, "OnPageFinished", url);
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("OnPageStarted")
@Description("")
String OnPageStartedEvents();
```

```
@DefaultMessage("OnPageFinished")
@Description("")
String OnPageFinishedEvents();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
OnPageStartedEvents = 开始加载网页
OnPageFinishedEvents = 完成网页加载
```

编译、运行系统后，在工作面板界面，可以看到增加了“完成网页加载”和“开始加载网页”两个事件响应函数，如图 13-3 所示。

使用示例代码如图 13-4 所示。



图 13-3　增加函数



图 13-4　代码

（这两个函数把正在加载网页的网址作为参数传递，也可以提示用户正在加载网页的网址。）

实现效果如图 13-5 所示。



图 13-5　加载网页

第 14 章　播放器的定制

14.1　音频播放器（Player）的定制

14.1.1　增加权限

音频播放器的源码文件为 Player.java，其中申请了如下权限：

```
@UsesPermissions(permissionNames = "android.permission.VIBRATE, android.permission.INTERNET")
```

但没有申请读取存储空间的权限，导致如果应用中没有使用其他可以读取存储空间的组件，只使用音频播放器组件的时候，音频播放器无法读取存储空间的文件。现增加读取存储空间的权限，解决此问题，代码如下：

```
@UsesPermissions(permissionNames = "android.permission.VIBRATE, android.permission.INTERNET,
android.permission.READ_EXTERNAL_STORAGE")
```

14.1.2　增加获取数据函数

App Inventor 2 在视频播放器中提供了获取文件时长的函数，但音频播放器中没有提供，现增加此函数。代码如下：

```
@SimpleFunction(description = "获取文件的时长")
public int GetDuration() {
  if (playerState == State.PREPARED || playerState == State.PLAYING || playerState == State.PAUSED_
  BY_USER) {
        return player.getDuration();
    } else {
      return 0;
    }
}
```

同时也增加一个获取当前播放时长的函数：

```
@SimpleFunction(description = "获取当前播放时长")
public int GetCurrentPosition() {
      return player.getCurrentPosition();
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("GetCurrentPosition")
@Description("")
String GetCurrentPositionMethods();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
GetCurrentPositionMethods = 获取当前播放时长
```

GetDuration 函数的名称和视频播放器中的函数名称相同，属于系统原有的，就不需要在 OdeMessages.java 和 OdeMessages_zh_CN.properties 中增加声明和字符串。

编译、运行系统后，在工作面板中，可以看到增加了“获取当前播放时长”和“求时长”两个新函数，如图 14-1 所示。



图 14-1　新增功能函数 1

14.1.3　增加播放函数

App Inventor 2 在视频播放器中提供了跳到指定的时长处的函数，但音频播放器中没有提供，现增加此函数。代码如下：

```
@SimpleFunction(description = "跳到指定的时长处")
public void SeekTo(int ms) {
   if (ms < 0) {
      ms = 0;
   }

   pause();
   player.seekTo(ms);
   Start();
}
```

SeekTo 函数的名称和视频播放器中的函数名称相同，属于系统原有的，就不需要在 OdeMessages.java 和 OdeMessages_zh_CN.properties 中增加声明和字符串。

编译、运行系统后，在工作面板中，可以看到增加了“搜寻”函数，如图 14-2 所示。



图 14-2　新增功能函数 2

14.2　视频播放器（VideoPlayer）的定制

视频播放器的源码文件为 VideoPlayer.java，其中申请了如下权限：

```
@UsesPermissions(permissionNames = "android.permission.INTERNET")
```

但没有申请读取存储空间的权限，导致如果应用中没有使用其他可以读取存储空间的组件，只使用视频播放器组件的时候，视频播放器无法读取存储空间的文件。现增加读取存储空间的权限，解决此问题，代码如下：

```
@UsesPermissions(permissionNames = "android.permission.INTERNET, android.permission.READ_
EXTERNAL_STORAGE")
```

第 15 章　画布（Canvas）的定制

App Inventor 2 的画布提供了绘制图形的功能，但能绘制的图形比较少，且功能比较单一；而 Android 系统本身的绘图功能是非常强大的，可以利用 Android 系统提供的函数增强画布的功能。

画布的源码文件为 Canvas.java，在其中增加以下代码。

15.1　增加画图函数

具体的代码如下：

```
//画圆弧
@SimpleFunction
public void DrawArc(int left, int top, int right, int bottom, float startAngle, float sweepAngle, boolean
useCenter, boolean fill) {
        float correctedLeft = left * $form().deviceDensity();
        float correctedTop = top * $form().deviceDensity();
        float correctedRight = right * $form().deviceDensity();
        float correctedBottom = bottom * $form().deviceDensity();

        Paint p = new Paint(paint);
        p.setStyle(fill ? Paint.Style.FILL : Paint.Style.STROKE);
        view.canvas.drawArc(correctedLeft, correctedTop, correctedRight, correctedBottom, startAngle,
        sweepAngle, useCenter, p);
        view.invalidate();
}

//画椭圆
@SimpleFunction
public void DrawOval(int left, int top, int right, int bottom, boolean fill) {
        float correctedLeft = left * $form().deviceDensity();
        float correctedTop = top * $form().deviceDensity();
        float correctedRight = right * $form().deviceDensity();
        float correctedBottom = bottom * $form().deviceDensity();

        Paint p = new Paint(paint);
        p.setStyle(fill ? Paint.Style.FILL : Paint.Style.STROKE);
        view.canvas.drawOval(correctedLeft, correctedTop, correctedRight, correctedBottom, p);
        view.invalidate();
```

```
    }

    //画矩形
    @SimpleFunction
    public void DrawRect(int left, int top, int right, int bottom, boolean fill) {
        float correctedLeft = left * $form().deviceDensity();
        float correctedTop = top * $form().deviceDensity();
        float correctedRight = right * $form().deviceDensity();
        float correctedBottom = bottom * $form().deviceDensity();

        Paint p = new Paint(paint);
        p.setStyle(fill ? Paint.Style.FILL : Paint.Style.STROKE);
        view.canvas.drawRect(correctedLeft, correctedTop, correctedRight, correctedBottom, p);
        view.invalidate();
    }

    //画圆角矩形
    @SimpleFunction
    public void DrawRoundRect(int left, int top, int right, int bottom, int rx, int ry, boolean fill) {
        float correctedLeft = left * $form().deviceDensity();
        float correctedTop = top * $form().deviceDensity();
        float correctedRight = right * $form().deviceDensity();
        float correctedBottom = bottom * $form().deviceDensity();
        float correctedRx = rx * $form().deviceDensity();
        float correctedRy = ry * $form().deviceDensity();

        Paint p = new Paint(paint);
        p.setStyle(fill ? Paint.Style.FILL : Paint.Style.STROKE);
        view.canvas.drawRoundRect(correctedLeft, correctedTop, correctedRight, correctedBottom,
        correctedRx, correctedRy, p);
        view.invalidate();
    }
```

15.2 增加绘制路径函数

在实际生活中，要绘制的图形是各种各样的，为了增加绘制各种图形的灵活性，Android 系统还提供了一个类：Path，用于绘制各种形状的路径，也可以理解为绘制各种形状图形的轮廓。

增加如下代码：

```
    import android.graphics.Path;

    //增加 Path 类变量
    private final Path path;
```

```
//在构造函数中初始化变量
public Canvas(ComponentContainer container) {
    …
    path = new Path();
    …
}

//设置路径的起点
@SimpleFunction
public void PathStart(int x, int y) {
    float correctedX = x * $form().deviceDensity();
    float correctedY = y * $form().deviceDensity();

    path.moveTo(correctedX, correctedY);
}

//设置路径的节点
@SimpleFunction
public void PathlineTo(int x, int y) {
    float correctedX = x * $form().deviceDensity();
    float correctedY = y * $form().deviceDensity();

    path.lineTo(correctedX, correctedY);
}

//设置路径为三次贝塞尔曲线
@SimpleFunction
public void PathCubicTo(int x1, int y1, int x2, int y2, int x3, int y3) {
    float correctedX1 = x1 * $form().deviceDensity();
    float correctedY1 = y1 * $form().deviceDensity();
    float correctedX2 = x2 * $form().deviceDensity();
    float correctedY2 = y2 * $form().deviceDensity();
    float correctedX3 = x3 * $form().deviceDensity();
    float correctedY3 = y3 * $form().deviceDensity();

    path.cubicTo(correctedX1, correctedY1, correctedX2, correctedY2, correctedX3, correctedY3);
}

//设置路径为二次贝塞尔曲线
@SimpleFunction
public void PathQuadTo(int x1, int y1, int x2, int y2) {
    float correctedX1 = x1 * $form().deviceDensity();
    float correctedY1 = y1 * $form().deviceDensity();
    float correctedX2 = x2 * $form().deviceDensity();
    float correctedY2 = y2 * $form().deviceDensity();
```

```
        path.quadTo(correctedX1, correctedY1, correctedX2, correctedY2);
    }

    //设置为封闭路径
    @SimpleFunction
    public void PathClose() {
        path.close();
    }

    //绘制路径
    @SimpleFunction
    public void DrawPath(boolean fill) {
        Paint p = new Paint(paint);
        p.setStyle(fill ? Paint.Style.FILL : Paint.Style.STROKE);
        view.canvas.drawPath(path, p);
        view.invalidate();
    }

    //清空路径
    @SimpleFunction
    public void PathReset() {
        path.reset();
    }
```

15.3 增加滑动事件响应函数

App Inventor 2 提供了手指在屏幕上按下和手松开时的事件响应函数，但没有提供手指在屏幕上滑动时的事件响应函数，这样就无法实现诸如手指在屏幕上滑动的同时，绘制图形的功能。增加此响应函数的代码如下：

```
    @SimpleEvent
    public void TouchMove(float x, float y) {
        EventDispatcher.dispatchEvent(this, "TouchMove", x, y);
    }

    void parse(MotionEvent event) {
            int width = Width();
            int height = Height();

            // Coordinates less than 0 can be returned if a move begins within a
            // view and ends outside of it.   Because negative coordinates would
            // probably confuse the user (as they did me) and would not be useful,
            // we replace any negative values with zero.
            float x = Math.max(0, (int) event.getX() / $form().deviceDensity());
            float y = Math.max(0, (int) event.getY() / $form().deviceDensity());
```

```
        …
        switch (event.getAction()) {
            …
            case MotionEvent.ACTION_MOVE:
                …
                //在此接收系统发出的移动事件消息
                TouchMove(x, y);
                …
        }
    }
```

15.4 设置函数参数的默认值

在 Drawer.js 文件的 component_method 数组中，增加如下代码，把上述函数的布尔型参数的默认值设置为 TRUE。

```
{matchingMutatorAttributes:{component_type:"Canvas", method_name:"DrawArc"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
          return '' +
            '<xml>' +
            '<block type="component_method">' +
            //mutator generator
            Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
            '<value name="ARG6"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '<value name="ARG7"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '</block>' +
            '</xml>';}},

          {matchingMutatorAttributes:{component_type:"Canvas", method_name:"DrawOval"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
          return '' +
            '<xml>' +
            '<block type="component_method">' +
            //mutator generator
            Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
            '<value name="ARG4"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '</block>' +
            '</xml>';}},

        {matchingMutatorAttributes:{component_type:"Canvas", method_name:"DrawRect"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
          return '' +
```

```
            '<xml>' +
            '<block type="component_method">' +
            //mutator generator
            Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
            '<value name="ARG4"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '</block>' +
            '</xml>';}},

        {matchingMutatorAttributes:{component_type:"Canvas", method_name:"DrawRoundRect"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
          return '' +
            '<xml>' +
            '<block type="component_method">' +
            //mutator generator
            Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
            '<value name="ARG6"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '</block>' +
            '</xml>';}},

        {matchingMutatorAttributes:{component_type:"Canvas", method_name:"DrawPath"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
          return '' +
            '<xml>' +
            '<block type="component_method">' +
            //mutator generator
            Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
            '<value name="ARG0"><block type="logic_boolean"><title name="BOOL">TRUE</title>
            </block></value>' +
            '</block>' +
            '</xml>';}},
```

15.5 增加声明和中文字符串

在 OdeMessages.java 中增加如下函数和参数的声明：

```
  @DefaultMessage("DrawArc")
  @Description("")
  String DrawArcMethods();

  @DefaultMessage("DrawOval")
  @Description("")
  String DrawOvalMethods();
```

```
@DefaultMessage("DrawRoundRect")
@Description("")
String DrawRoundRectMethods();

@DefaultMessage("PathStart")
@Description("")
String PathStartMethods();

@DefaultMessage("PathlineTo")
@Description("")
String PathlineToMethods();

@DefaultMessage("PathCubicTo")
@Description("")
String PathCubicToMethods();

@DefaultMessage("PathQuadTo")
@Description("")
String PathQuadToMethods();

@DefaultMessage("PathClose")
@Description("")
String PathCloseMethods();

@DefaultMessage("DrawPath")
@Description("")
String DrawPathMethods();

@DefaultMessage("PathReset")
@Description("")
String PathResetMethods();

@DefaultMessage("TouchMove")
@Description("")
String TouchMoveEvents();

@DefaultMessage("x3")
@Description("")
String x3Params();

@DefaultMessage("startAngle")
@Description("")
String startAngleParams();

@DefaultMessage("sweepAngle")
@Description("")
```

```
String sweepAngleParams();

@DefaultMessage("useCenter")
@Description("")
String useCenterParams();

@DefaultMessage("r x")
@Description("")
String rxParams();

@DefaultMessage("r y")
@Description("")
String ryParams();
```

在 OdeMessages_zh_CN.properties 中增加中文字符串：

```
DrawArcMethods = 画弧形
DrawOvalMethods = 画椭圆
DrawRoundRectMethods = 画圆角矩形
PathStartMethods = 路径起点
PathlineToMethods = 路径节点
PathCubicToMethods = 三次贝塞尔曲线
PathQuadToMethods = 二次贝塞尔曲线
PathCloseMethods = 封闭路径
DrawPathMethods = 绘制路径
PathResetMethods = 清空路径
TouchMoveEvents = 手指在画布上移动

angleParams = 角度
startAngleParams = 开始角度
sweepAngleParams = 弧形的角度
useCenterParams = 是否封闭弧形
rxParams = x 轴方向圆角半径
ryParams = y 轴方向圆角半径
```

编译、运行系统后，在工作面板中，可以看到增加了如图 15-1～图 15-6 所示函数。

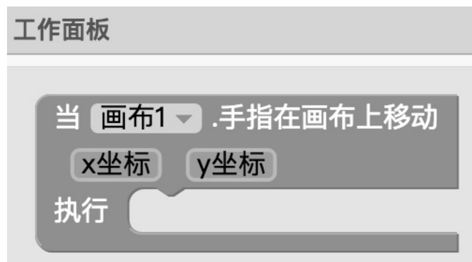


图 15-1　新增事件响应函数



图 15-2　新增功能函数 1



图 15-3　新增功能函数 2



图 15-4　新增功能函数 3

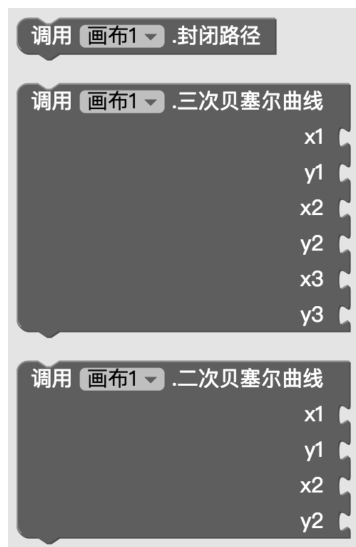


图 15-5　新增功能函数 4



图 15-6　新增功能函数 5

15.6　路径函数使用示例

如果要画一个三角形，并且三角形内部是填充的，使用常规的画图函数是无法实现的，这就需要用路径函数来实现。

具体代码如图 15-7 所示。



图 15-7　代码

绘制的图形如图 15-8 所示。



图 15-8　填充的三角形

在调用“绘制路径”函数后，要调用“清空路径”函数清除路径数据，否则下次绘制的时候，会重复之前的绘制。

第 16 章　文件管理器（File）的定制

16.1　增加文件处理函数

App Inventor 2 内置的文件管理器没有修改文件名和获取文件大小两个常用功能，现修改源码实现这两个功能。

文件管理器的源码文件是 File.java，在其中增加两个函数：

```
@SimpleFunction(description = "修改文件名")
public void Rename(String oldFileName, String newFileName) {
   java.io.File file = new java.io.File(oldFileName);

   if(!file.exists()){
         try {
            file.createNewFile();
         } catch (IOException e) {
               form.dispatchErrorOccurredEvent(File.this, "RenameFile",
                     ErrorMessages.ERROR_CANNOT_CREATE_FILE, oldFileName);

               return;
         }
   }

   file.renameTo(new java.io.File(newFileName));
}

@SimpleFunction(description = "获取文件大小")
public long GetFileSize(String fileName) {
   java.io.File file = new java.io.File(fileName);

   if(file.exists()){
         return file.length();
   }else{
      return 0;
   }
}
```

在 OdeMessages.java 中增加函数和函数参数的声明：

```
//函数声明
```

```
@DefaultMessage("GetFileSize")
@Description("")
String GetFileSizeMethods();

@DefaultMessage("Rename")
@Description("")
String RenameMethods();
//函数参数声明，fileName 参数的声明是系统原先就有的
@DefaultMessage("fileName")
@Description("")
String fileNameParams();

@DefaultMessage("oldFileName")
@Description("")
String oldFileNameParams();

@DefaultMessage("newFileName")
@Description("")
String newFileNameParams();
```

在 OdeMessages_zh_CN.properties 中增加函数和参数的中文字符串：

```
fileNameParams = 文件名
oldFileNameParams = 旧文件名
newFileNameParams = 新文件名
RenameMethods = 修改文件名
GetFileSizeMethods = 获取文件大小
```

编译、运行系统后，在工作面板中，可以看到增加了“获取文件大小”和“读取文件名”两个函数，如图 16-1 所示。

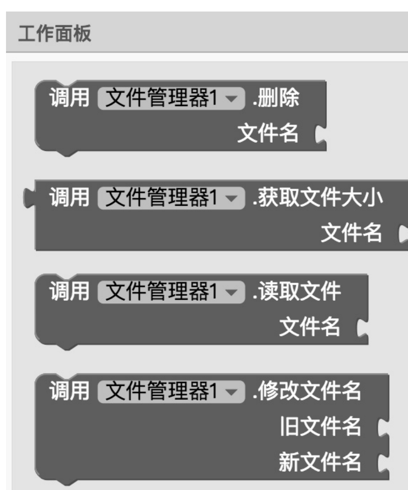


图 16-1　新增功能函数 1

16.2 增加获取存储区信息的函数

增加两个函数用于获取存储空间根目录和剩余存储空间大小，代码如下：

```
@SimpleFunction(description = "获取存储空间根目录")
public String GetExternalStorageDirectory() {
    return Environment.getExternalStorageDirectory().getPath();
}

//返回值的单位是 byte
@SimpleFunction(description = "获取剩余存储空间大小")
public long GetUsableSpace() {
    return Environment.getExternalStorageDirectory().getUsableSpace();
}
```

在 OdeMessages.java 中增加函数的声明：

```
@DefaultMessage("GetExternalStorageDirectory")
@Description("")
String GetExternalStorageDirectoryMethods();

@DefaultMessage("GetUsableSpace")
@Description("")
String GetUsableSpaceMethods();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
GetExternalStorageDirectoryMethods = 获取存储空间根目录
GetUsableSpaceMethods = 获取剩余存储空间大小
```

编译、运行系统后，在工作面板中，可以看到增加了“获取存储空间根目录”和“获取剩余存储空间大小”两个函数，如图 16-2 所示。



图 16-2　新增功能函数 2

第 17 章　微数据库组件（TinyDB）的定制

17.1　存储功能介绍

微数据库组件的源码文件是 TinyDB.java，创建数据库的代码如下：

```
public TinyDB(ComponentContainer container) {
    super(container.$form());
    context = (Context) container.$context();
    sharedPreferences = context.getSharedPreferences("TinyDB1", Context.MODE_PRIVATE);
}
```

从代码中可以看出，微数据库存储是利用了 Android 系统的 SharedPreferences 框架，保存和检索原始数据类型的永久性键值对。可以使用 SharedPreferences 来保存任何原始数据：布尔值、浮点值、整型值、长整型和字符串。即使应用停止运行了，数据也会被保留。

这种框架是把数据保存在 xml 文件中，getSharedPreferences 函数中的第一个参数就是 xml 的文件名，第二个参数标识对这个文件的操作模式，MODE_PRIVATE 标识此文件只能被创建它的应用，或与此应用有相同 use ID 的应用进行读写操作。还有以下几种模式：MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE 和 MODE_MULTI_PROCESS。但通常都使用 MODE_PRIVATE 这种模式。

创建的 xml 文件存储在 Android 设备的如下路径：

```
/data/data/应用的包名/shared_prefs
```

在应用中向微数据库中存入数据的代码如图 17-1 所示。

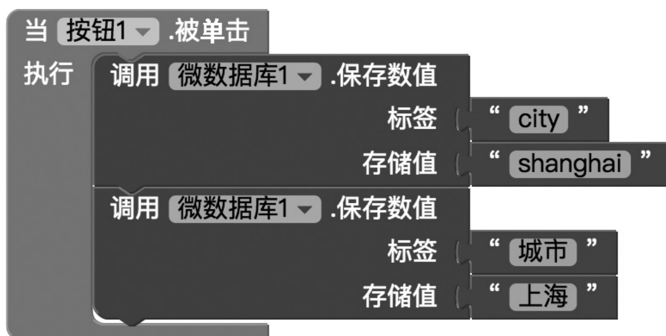


图 17-1　代码 1

利用 Android 系统的 adb 和 shell 命令，在 Android 模拟器的如下路径可以看到创建的 xml 文件：

```
root@generic_x86:/data/data/appinventor.ai_test.demo/shared_prefs # ls
TinyDB1.xml
```

appinventor.ai_test.demo 是应用的包名

TinyDB1.xml 就是创建的 xml 文件，其中的内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="城市">&quot;上海&quot;</string>
    <string name="city">&quot;shanghai&quot;</string>
</map>
```

17.2　增加设置数据文件名称和数据加密功能

通常在应用中，数据可能需要存储在不同的 xml 文件中，但 App Inventor 2 默认所有的数据都存储在 TinyDB1.xml 这个文件中，使用不灵活。

现修改代码，增加以下函数，使开发人员可以把数据存储在不同的 xml 文件中，其中相比原有函数多出的 name 参数，就是 xml 文件的名称。

有时为了安全起见，保存到本地的数据还需要进行加密处理，StoreValueNew 函数的最后一个参数标识是否对数据进行 MD5 加密。

增加如下代码：

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

@SimpleFunction
public void StoreValueNew(final String name, final String tag, final Object valueToStore, final boolean
bIsEncryption) {

    sharedPreferencesNew = context.getSharedPreferences(name, Context.MODE_PRIVATE);

    final SharedPreferences.Editor sharedPrefsEditor = sharedPreferencesNew.edit();
    try {
      String jsonString = JsonUtil.getJsonRepresentation(valueToStore);

      if (!bIsEncryption){
        sharedPrefsEditor.putString(tag, jsonString);
      } else {
        String encryptedString = makeMD5Hash(jsonString);
        sharedPrefsEditor.putString(tag, encryptedString);
      }
```

```
            sharedPrefsEditor.commit();
        } catch (JSONException e) {
            throw new YailRuntimeError("Value failed to convert to JSON.", "JSON Creation Error.");
        }
    }

    @SimpleFunction
    public Object GetValueNew(final String name, final String tag, final Object valueIfTagNotThere) {
        try {
            sharedPreferencesNew = context.getSharedPreferences(name, Context.MODE_PRIVATE);

            String value = sharedPreferencesNew.getString(tag, "");
            // If there's no entry with tag as a key then return the empty string.
            // was return (value.length() == 0) ? "" : JsonUtil.getObjectFromJson(value);
            return (value.length() == 0) ? valueIfTagNotThere : JsonUtil.getObjectFromJson(value);
        } catch (JSONException e) {
            throw new YailRuntimeError("Value failed to convert from JSON.", "JSON Creation Error.");
        }
    }

    @SimpleFunction
    public Object GetTagsNew(final String name) {
        sharedPreferencesNew = context.getSharedPreferences(name, Context.MODE_PRIVATE);

        List<String> keyList = new ArrayList<String>();
        Map<String,?> keyValues = sharedPreferencesNew.getAll();
        // here is the simple way to get keys
        keyList.addAll(keyValues.keySet());
        java.util.Collections.sort(keyList);
        return keyList;
    }

    @SimpleFunction
    public void ClearAllNew(final String name) {
        sharedPreferencesNew = context.getSharedPreferences(name, Context.MODE_PRIVATE);

        final SharedPreferences.Editor sharedPrefsEditor = sharedPreferencesNew.edit();
        sharedPrefsEditor.clear();
        sharedPrefsEditor.commit();
    }

    @SimpleFunction
    public void ClearTagNew(final String name, final String tag) {
        sharedPreferencesNew = context.getSharedPreferences(name, Context.MODE_PRIVATE);
```

```
        final SharedPreferences.Editor sharedPrefsEditor = sharedPreferencesNew.edit();
        sharedPrefsEditor.remove(tag);
        sharedPrefsEditor.commit();
}

private String makeMD5Hash(String key) {
      String cacheKey;
      try {
        final MessageDigest mDigest = MessageDigest.getInstance("MD5");
        mDigest.reset();
        mDigest.update(key.getBytes());
        cacheKey = bytesToHexString(mDigest.digest());
      } catch (NoSuchAlgorithmException e) {
        cacheKey = String.valueOf(key.hashCode());
      }
      return cacheKey;
}

private String bytesToHexString(byte[] bytes) {
    StringBuilder sb = new StringBuilder();
    for (byte value : bytes) {
        String hex = Integer.toHexString(0xFF & value);
        if (hex.length() == 1) {
          sb.append('0');
        }
        sb.append(hex);
    }
    return sb.toString();
}
```

在 OdeMessages.java 中增加函数和参数声明：

```
@DefaultMessage("GetValueNew")
@Description("")
String GetValueNewMethods();

@DefaultMessage("StoreValueNew")
@Description("")
String StoreValueNewMethods();

@DefaultMessage("ClearAllNew")
@Description("")
String ClearAllNewMethods();

@DefaultMessage("ClearTagNew")
@Description("")
String ClearTagNewMethods();
```

```
@DefaultMessage("GetTagsNew")
@Description("")
String GetTagsNewMethods();

//参数声明
@DefaultMessage("bIsEncryption")
@Description("")
String bIsEncryptionParams();
```

在 OdeMessages_zh_CN.properties 中增加函数和参数的中文字符串：

```
GetValueNewMethods = 获取数值
StoreValueNewMethods = 保存数值
ClearAllNewMethods = 清除所有数据
ClearTagNewMethods = 清除标签数据
GetTagsNewMethods = 获取标签数据

bIsEncryptionParams = 是否
```

App Inventor 2 提供的“获取数值”函数，如果输入的标签不存在，则返回空字符串，如图 17-2 所示。新增的“获取数值”函数，同样也能实现此功能。

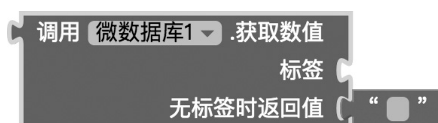


图 17-2 原有“获取数值”函数

StoreValueNew 函数的最后一个参数值默认为 false，不对数据加密。

需要修改 drawer.js 文件，实现对参数默认赋值，代码如下：

```
component_method: [
…
{matchingMutatorAttributes:{component_type:"TinyDB", method_name:"StoreValueNew"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
            return '' +
              '<xml>' +
              '<block type="component_method">' +
              //mutator generator
             Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
              '<value name="ARG3"><block type="logic_boolean"><title name="BOOL">FALSE</title>
              </block></value>' +
              '</block>' +
              '</xml>';}},

        {matchingMutatorAttributes:{component_type:"TinyDB", method_name:"GetValueNew"},
        mutatorXMLStringFunction: function(mutatorAttributes) {
            return '' +
              '<xml>' +
```

```
                '<block type="component_method">' +
                //mutator generator
                Blockly.Drawer.mutatorAttributesToXMLString(mutatorAttributes) +
                '<value name="ARG2"><block type="text"><title name="TEXT"></title></block></value>' +
                '</block>' +
                '</xml>';}},
        …
        ]
```

编译、运行系统后，在工作面板中，可以看到增加了 5 个新函数，这 5 个函数比原有函数多了参数：名称，用于处理不同的 xml 文件，如图 17-3～图 17-5 所示。



图 17-3　新增功能函数 1



图 17-4　新增功能函数 2



图 17-5　新增功能函数 3

17.3 使用示例

保存数据的代码如图 17-6 所示。



图 17-6 代码 2

执行完后，在如下路径里可以看到创建的两个 xml 文件：

```
root@generic_x86:/data/data/appinventor.ai_test.demo/shared_prefs # ls
man.xml
woman.xml
```

man.xml 的内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="姓名">&quot;王刚&quot;</string>
</map>
```

woman.xml 的内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="姓名">&quot;王芳&quot;</string>
</map>
```

读取文件内容的示例代码如图 17-7 所示。



图 17-7 代码 3

执行结果如图 17-8 所示。



图 17-8　执行结果 1

数据加密示例代码如图 17-9 所示。

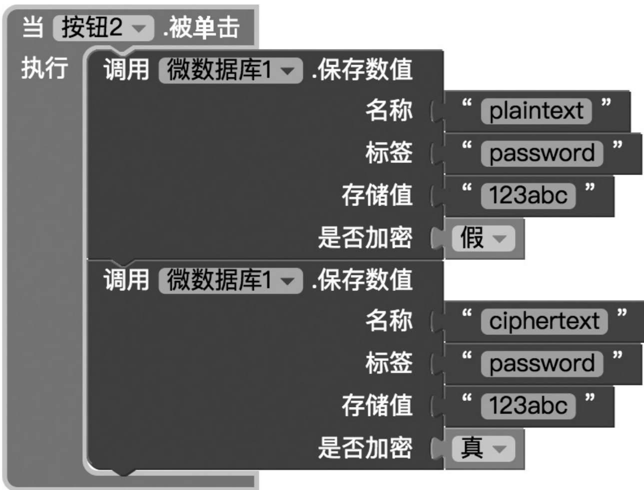


图 17-9　代码 4

代码执行完后，会生成两个文件，plaintext.xml 的内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">&quot;123abc&quot;</string>
</map>
```

ciphertext.xml 的内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">6def699c15f62b9d814ee315b8ce9ac4</string>
</map>
```

通过使用 MD5 加密，把明文字符串“123abc”转成了一个 32 位长度的字符串“6def699c15f62b9d814ee315b8ce9ac4”。

读取数据的代码如图 17-10 所示。



图 17-10　代码 5

执行结果如图 17-11 所示。

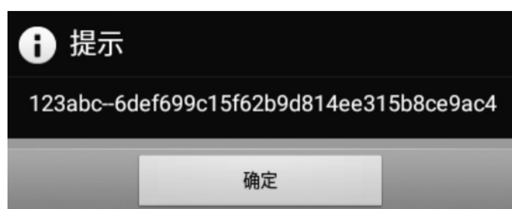


图 17-11　执行结果 2

第 18 章　Web 客户端的定制

18.1　增加 PATCH 请求函数

Web 客户端组件给开发人员提供了 GET、POST、PUT 和 DELETE 请求的功能函数，但没有提供 PATCH 请求的功能函数；PATCH 请求用于资源的部分内容的更新，例如，更新某一个字段，也属于比较常用的请求。

Web 客户端组件的源码文件是 Web.java，在其中增加如下代码：

```
@SimpleFunction(description = "Performs an HTTP PATCH request using the Url property and " +
    "the specified text.<br>" +
    "The characters of the text are encoded using UTF-8 encoding.<br>" +
    "If the SaveResponse property is true, the response will be saved in a file and the " +
    "GotFile event will be triggered. The responseFileName property can be used to specify " +
    "the name of the file.<br>" +
    "If the SaveResponse property is false, the GotText event will be triggered.")
public void PatchText(final String text) {
requestTextImpl(text, "UTF-8", "PatchText", "PATCH");
}

@SimpleFunction(description = "Performs an HTTP PATCH request using the Url property and " +
    "the specified text.<br>" +
    "The characters of the text are encoded using the given encoding.<br>" +
    "If the SaveResponse property is true, the response will be saved in a file and the " +
    "GotFile event will be triggered. The ResponseFileName property can be used to specify " +
    "the name of the file.<br>" +
    "If the SaveResponse property is false, the GotText event will be triggered.")
public void PatchTextWithEncoding(final String text, final String encoding) {
requestTextImpl(text, encoding, "PatchTextWithEncoding", "PATCH");
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("PatchText")
@Description("")
String PatchTextMethods();
```

```
@DefaultMessage("PatchTextWithEncoding")
@Description("")
String PatchTextWithEncodingMethods();
```

在 OdeMessages_zh_CN.properties 中增加函数的中文字符串：

```
PatchTextMethods = 执行 PATCH 文本请求
PatchTextWithEncodingMethods = 执行 PATCH 编码文本请求
```

编译、运行系统后，在工作面板中，可以看到增加了“执行 PATCH 文本请求”和“执行 PATCH 编码文本请求”两个功能函数，如图 18-1 所示。



图 18-1　Web 客户端新增函数

18.2　增加设置 JSON 格式数据的属性

目前应用和服务器间传输数据大多采用 JSON 格式的数据，App Inventor 2 提供了解析 JSON 格式数据的功能函数，却没有提供向服务器传递 JSON 格式数据的功能。

在使用“执行 POST 文本请求”函数向服务器传递数据时，利用抓包工具，可以看到如图 18-2 所示的数据。

```
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; Android SDK built for x86 Build/KK)
Host: restapi.amap.com
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
```

图 18-2　数据

其中 Content-Type 的值就是用于服务器指示传递的数据格式类型，默认不是 JSON 格式，即使传递给服务器的数据是 JSON 格式，但服务器根据 Content-Type 的值按非 JSON 格式数据解析，导致解析出错。如果要传递 JSON 格式数据，就需要设置 Content-Type 的值。

在 Web.java 中增加如下代码：

```
//标识是否使用 JSON 格式数据
private static boolean bIsJSON;

private static HttpURLConnection openConnection(CapturedProperties webProps, String httpVerb)
throws IOException, ClassCastException, ProtocolException {

    HttpURLConnection connection = (HttpURLConnection) webProps.url.openConnection();
    …
    //如果使用 JSON 格式数据，设置 Content-Type 的值为"application/json"，否则设置成默认值
    if (bIsJSON){
        connection.setRequestProperty("Content-Type", "application/json");
          } else {
         connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
     }
    …
    return connection;
}

@SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "设置是否传递 JSON 格式
的数据给服务器")
public boolean IsJSON() {
    return bIsJSON;
}

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, default
Value = "False")
@SimpleProperty
public void IsJSON(boolean bIsJSON) {
    this.bIsJSON = bIsJSON;
}
```

在 OdeMessages.java 中增加函数声明：

```
@DefaultMessage("IsJSON")
@Description("")
String IsJSONProperties();
```

在 OdeMessages_zh_CN.properties 中增加属性的中文字符串：

```
IsJSONProperties = 使用 JSON 格式数据
```

编译、运行系统后，在组件属性中，可以看到增加了“使用 JSON 格式数据”的属性，如图 18-3 所示。

在工作面板中，可以看到增加了此属性的 gette 和 setter 函数，如图 18-4 所示。

图 18-3　Web 客户端新增属性

图 18-4　Web 客户端新增属性函数

第 19 章　插件（Extension）开发

为了扩展 App Inventor 2 的功能，除了修改 App Inventor 2 原有的组件外，还可以自己开发功能插件，方便开发应用使用。

19.1　加密功能插件

19.1.1　插件的实现

之前提到过的加密功能，就可以做成一个插件，使用起来更灵活。

首先需要建立存放插件源码的文件夹：

```
/appinventor/components/src/com/qz/extensions
```

然后创建源码文件 Encryption.java，其中的代码如下：

```
//插件的包名，通常是三段式 com. + 功能描述. + extension
package com.encryption.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

@DesignerComponent(
    //插件的版本号
    version = Encryption.VERSION,
    //插件的说明文字
    description = "功能：数据加密  开发者：QZ",
    //如果在网上有插件的说明文字，可以在此设置网址
    helpUrl = "http://www.baidu.com",
    //设置组件的类型为插件
    category = ComponentCategory.EXTENSION,
```

```
        //插件是否可见，由于目前 App Inventor 2 只能开发和使用不可见的插件，因此这里只能设置
        //成 true
        nonVisible = true,
        //插件的图标，在此使用 App Inventor 2 提供的图标
        iconName = "images/extension.png")

//external 的值为 true，设置此组件为外部组件，编译器会将此 java 文件打包生成一个 aix 文件
@SimpleObject(external = true)
public class Encryption extends AndroidNonvisibleComponent implements Component {
        public static final int VERSION = 1;
        private ComponentContainer container;
        private Context context;
        private static final String LOG_TAG = "Encryption";
        public Encryption(ComponentContainer container) {
                super(container.$form());
                this.container = container;
                context = (Context) container.$context();
                Log.d(LOG_TAG, "Encryption Created" );
        }

        //此插件对外提供的函数
        @SimpleFunction(description = "对数据进行 MD5 加密")
        public String makeMD5Hash(String srcStr) {
        String MD5Str;
        try {
            final MessageDigest mDigest = MessageDigest.getInstance("MD5");
            mDigest.reset();
            mDigest.update(srcStr.getBytes());
            MD5Str = bytesToHexString(mDigest.digest());
        } catch (NoSuchAlgorithmException e) {
            MD5Str = String.valueOf(srcStr.hashCode());
        }
        return MD5Str;
    }

    private String bytesToHexString(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte value : bytes) {
            String hex = Integer.toHexString(0xFF & value);
            if (hex.length() == 1) {
                sb.append('0');
            }
            sb.append(hex);
        }
```

```
            return sb.toString();
        }
    }
```

19.1.2　插件的编译

在 appinventor 目录下执行“ant extensions”命令即可生成相应的插件：com.encryption.extension.aix，生成的文件存放路径如下：

```
appinventor-sources/appinventor/components/build/extensions/com.encryption.extension.aix
```

19.1.3　插件的导入

使用插件，首先需要导入插件，具体步骤如下：

1）单击图 19-1 所示组件面板最下面的“Import extension”字符串标签，显示图 19-2 所示对话框。



图 19-1　组件面板

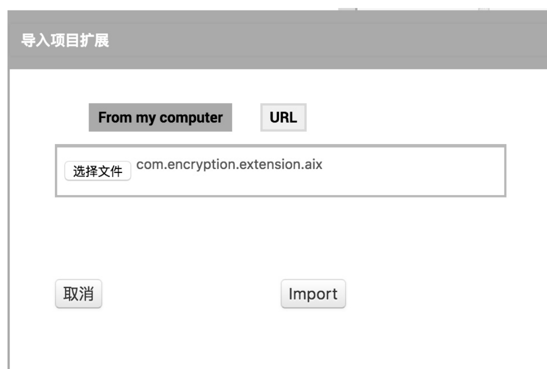


图 19-2　导入对话框

2）选择文件后，单击“Import”按钮，可以看到如图 19-3 所示的界面，组件面板底部多了个名为“Encryption”的插件。



图 19-3　插件被导入

3）单击问号图标，可以看到如图 19-4 所示 DesignerComponent 注解添加的说明文字。



图 19-4　插件说明

然后就可以像使用 App Inventor 2 自带的组件那样使用插件了。如图 19-5 所示可以把插件添加在组件列表中，且在工作面板中可以看到插件提供的函数，如图 19-6 所示。

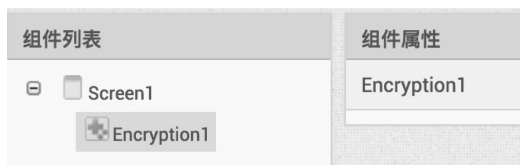


图 19-5　添加插件到组件列表



图 19-6　加密功能插件函数

19.1.4　插件的使用

使用示例代码如图 19-7 所示。



图 19-7　加密功能插件使用代码

因为是插件开发，没有修改 App Inventor 2 里的资源文件，所以即使 App Inventor 2 的系统语言设置为中文，插件的函数和参数名都是英文。

19.2　对话框插件

19.2.1　插件的实现

也可以对系统提供的组件进行插件开发，如对系统对话框组件进行插件开发，把 Notifier.java 文件名改为 QZNotifier.java，以便与原有的文件区别，然后复制到/appinventor/components/src/com/qz/extensions 中。

对源码做如下修改：

```
//package com.google.appinventor.components.runtime;
//修改包名
```

```
package com.notifier.extension;
import com.google.appinventor.components.runtime.*;

//修改 DesignerComponent 注解的相关内容
@DesignerComponent(version = QZNotifier.VERSION,
    category = ComponentCategory.EXTENSION,
    description = "功能：显示各类对话框    开发者：QZ",
    nonVisible = true,
    iconName = "images/extension.png")

//修改 SimpleObject 注解的相关内容
@SimpleObject(external = true)
//根据 java 语言规范，把类名和构造函数名称改成和文件名一致
public final class QZNotifier extends AndroidNonvisibleComponent implements Component {
    //增加插件的版本号变量
    public static final int VERSION = 1;
    …
    public QZNotifier (ComponentContainer container) {
        super(container.$form());
        activity = container.$context();
        handler = new Handler();
        progressDialog = null;
    }
    …
}
```

19.2.2 插件的导入

编译并按之前描述的步骤导入、添加插件到组件列表中后，在组件属性中看到有英文属性，如图 19-8 所示。



图 19-8　插件提供的属性

在工作面板中，看到有英文函数，如图 19-9 和图 19-10 所示。



图 19-9 对话框插件函数 1



图 19-10 对话框插件函数 2

因为是开发插件，没有修改 App Inventor 2 里的资源文件，所以即使 App Inventor 2 的系统语言设置为中文，新增的函数和属性名称都是英文。

19.3 获取设备和系统信息插件

此插件用于获取设备和设备上运行的系统的相关信息，包括设备型号、设备的 IMEI 号码、设备屏幕的宽度、设备屏幕的高度、屏幕的 DPI、系统的 API 级别、系统的版本、系统当前时间和当前运行应用的版本号等。

19.3.1 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 SystemInfo.java，其中的代码如下：

```
package com.systeminfo.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.runtime.*;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.telephony.TelephonyManager;
import android.util.DisplayMetrics;
```

```
import android.view.View;
import android.view.WindowManager;
import java.text.SimpleDateFormat;
import java.util.Date;

@DesignerComponent(version = SystemInfo.VERSION,
        description = "功能：获取设备和系统信息    开发者：QZ",
        helpUrl = "http://blog.csdn.net/xjbclz",
        category = ComponentCategory.EXTENSION,
        nonVisible = true,
        iconName = "images/extension.png")

@SimpleObject(external = true)
@UsesPermissions(permissionNames = "android.permission.READ_PHONE_STATE")
public class SystemInfo extends AndroidNonvisibleComponent {
        public static final int VERSION = 1;
        private ComponentContainer container;
        private Context context;

        private static final String LOG_TAG = "SystemInfo";
        public SystemInfo(ComponentContainer container) {
                super(container.$form());
                this.container = container;
                context = (Context) container.$context();

                Log.d(LOG_TAG, "SystemInfo Created" );
        }

        @SimpleFunction(description = "获取设备型号")
        public String GetModel() {
                return android.os.Build.MODEL;
        }

        @SimpleFunction(description = "获取设备的 IMEI 号")
        public String GetIMEI() {
                TelephonyManager tm = (TelephonyManager) context.getSystemService(Context. TELEPHONY_
                SERVICE);
                return tm.getDeviceId();
        }

        @SimpleFunction(description = "获取屏幕的宽度")
        public int GetWidth() {
                WindowManager wm = (WindowManager) context.getSystemService(Context.WINDOW_
                SERVICE);
                return wm.getDefaultDisplay().getWidth();
        }
```

```
@SimpleFunction(description = "获取屏幕的高度")
public int GetHeight() {
    WindowManager wm = (WindowManager) context.getSystemService(Context.WINDOW_
    SERVICE);
    return wm.getDefaultDisplay().getHeight();
}

@SimpleFunction(description = "获取屏幕的 DPI")
public int GetDPI() {
    return context.getResources().getDisplayMetrics().densityDpi;
}

@SimpleFunction(description = "获取系统的 API 级别")
public String GetSysAPI() {
    return android.os.Build.VERSION.SDK;
}

@SimpleFunction(description = "获取系统的版本")
public String GetSysVersion() {
    return android.os.Build.VERSION.RELEASE;
}

@SimpleFunction(description = "获取系统当前时间戳")
public long GetSysTimeStamp() {
    return System.currentTimeMillis();
}

@SimpleFunction(description = "获取系统当前时间")
public String GetSysTime() {
    long time=System.currentTimeMillis();
    Date date=new Date(time);
    SimpleDateFormat format=new SimpleDateFormat("yyyyMMddHHmmss");

    return format.format(date);
}

@SimpleFunction(description = "获取当前运行应用的版本")
public String GetAppVersion() {
    try {
        PackageManager pm = context.getPackageManager();
        PackageInfo packageInfo = pm.getPackageInfo(context.getPackageName(),Package
        Manager.GET_CONFIGURATIONS);

        return packageInfo.versionName;
    } catch (Exception e) {
```

```
                        e.printStackTrace();
                    }

                    return null;
                }
            }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-11 所示。



图 19-11　获取设备和系统信息插件函数

19.3.2　插件的使用

Android 系统有多个版本，各版本支持的功能也不完全一样，像前面描述过的实现对话框显示风格为 Material Design 风格的功能，只有在 Android5.0(API 21)及以上版本中是默认支持的；在使用这个功能时，就需要获取系统的版本，看其是否支持此功能。

示例代码如图 19-12 所示。



图 19-12　获取设备和系统信息插件使用代码 1

对话框的默认显示风格是非 Material Design 风格的，在 Screen 初始化的时候，获取当前系统的 API，当值大于等于 21 时，设置显示风格为 Material Design 风格。

有时在退出某个屏幕的时候，需要保存数据到文件中，以便后续使用，这类文件是App自动保存的，不需要用户输入文件名，但又需要保证文件名的唯一性，就可以使用保存文件时的系统当前时间作为文件名，代码如图 19-13 所示。



图 19-13　获取设备和系统信息插件使用代码 2

19.4　监测设备电量状态插件

在使用移动设备的时候，经常遇到电量不足的情况，如果应用要用到一些比较耗电的功能，如多媒体功能和上网功能等，最好要实时监测设备的电量状态，以便提供更好的用户体验。

Android 系统提供了系统广播，用于监测设备的电量状态，包括电量不足、电量充足和电量发生改变。具体的实现方式如下。

19.4.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 BatteryState.java，其中的代码如下：

```
package com.batterystate.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;

@DesignerComponent(version = BatteryState.VERSION,
    description = "功能：监测电量状态　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
```

```
        category = ComponentCategory.EXTENSION,
        nonVisible = true,
        iconName = "images/extension.png")

@SimpleObject(external = true)
//申请获取电池状态的系统权限
@UsesPermissions(permissionNames = "android.permission.BATTERY_STATS")
public class BatteryState extends AndroidNonvisibleComponent implements Component,
OnDestroyListener {
        public static final int VERSION = 1;
        private ComponentContainer container;
        private Context context;
        private final BatteryStateReceiver batteryStateReceiver;

        private static final String LOG_TAG = "BatteryState";
        public BatteryState(ComponentContainer container) {
                super(container.$form());
                this.container = container;
                context = (Context) container.$context();

                form.registerForOnDestroy(this);
                batteryStateReceiver = new BatteryStateReceiver();
                registerBatteryStateMonitor();

                Log.d(LOG_TAG, "BatteryState Created" );
        }

        //实现广播接收器，接收和处理系统广播
        private class BatteryStateReceiver extends BroadcastReceiver {
                public BatteryStateReceiver() {

                }

                @Override
                public void onReceive(Context context, Intent intent) {
                        String action = intent.getAction();
                        if(Intent.ACTION_BATTERY_LOW.equals(action)){
                                BatteryStateLow();
                        } else if (Intent.ACTION_BATTERY_OKAY.equals(action)) {
                                BatteryStateOK();
                        }else if (Intent.ACTION_BATTERY_CHANGED.equals(action)) {
                                // 获得当前电量
                                int currentLevel = intent.getExtras().getInt("level");
                                // 获得总电量
                                int total = intent.getExtras().getInt("scale");
```

```
                        int percent = currentLevel * 100 / total;
                        BatteryLevel(percent);
                    }
                }
            }

            //注册广播接收器，接收三个系统广播
            private void registerBatteryStateMonitor(){
                IntentFilter intentFilter = new IntentFilter();
                intentFilter.addAction(Intent.ACTION_BATTERY_LOW);
                intentFilter.addAction(Intent.ACTION_BATTERY_OKAY);
                intentFilter.addAction(Intent.ACTION_BATTERY_CHANGED);
                context.registerReceiver(batteryStateReceiver, intentFilter);
            }

            //注销广播接收器
            private void unregisterBatteryStateMonitor(){
                context.unregisterReceiver(batteryStateReceiver);
            }

            //销毁当前屏幕时，注销广播接收器
            @Override
            public void onDestroy() {
                unregisterBatteryStateMonitor();
            }

            //实现以下三个事件函数，供外部调用
            @SimpleEvent(description = "反馈电量不足")
            public void BatteryStateLow() {
                EventDispatcher.dispatchEvent(this, "BatteryStateLow");
            }

            @SimpleEvent(description = "反馈电量充足")
            public void BatteryStateOK() {
                EventDispatcher.dispatchEvent(this, "BatteryStateOK");
            }

            @SimpleEvent(description = "反馈电量百分比")
            public void BatteryLevel(int level) {
                EventDispatcher.dispatchEvent(this, "BatteryLevel", level);
            }

        }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-14 所示。



图 19-14　监测设备电量状态插件函数

19.4.2　插件的使用

使用示例代码如图 19-15 和图 19-16 所示。



图 19-15　监测设备电量状态插件使用代码 1



图 19-16　监测设备电量状态插件使用代码 2

上述代码实现了如下功能：

在播放音乐的时候，如监测到电量不足，显示提示框，提示用户“电量不足，是否暂停播放音乐?”。

在暂停后，如监测到电量充足，显示提示框，提示用户“电量充足，是否播放音乐?”。

19.5 获取和监测设备网络状态插件

在使用移动设备的时候，移动设备的网络状态经常发生变化，不同的网络状态，对于用户使用网络功能的影响非常大，所以如果应用使用了网络功能，同样要实时监听设备的网络状态，以便提供更好的用户体验。

Android 系统提供了系统广播，用于监测设备的网络状态，也可以利用 Connectivity Manager 类，获取设备的网络状态，包括是否连接网络，以及是 WIFI、2G、3G 还是 4G 状态下连接网络。具体的实现方式如下。

19.5.1 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 NetworkState.java，其中的代码如下：

```
package com.networkstate.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;

import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.telephony.TelephonyManager;

@DesignerComponent(version = NetworkState.VERSION,
    description = "功能：获取和监测网络状态   开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")
```

```
@SimpleObject(external = true)
//申请获取网络状态的系统权限
@UsesPermissions(permissionNames = "android.permission.ACCESS_NETWORK_STATE")
public class NetworkState extends AndroidNonvisibleComponent implements Component, On
DestroyListener {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;
    private final NetworkStateReceiver networkStateReceiver;
    private String networkState;

    private static final String LOG_TAG = "NetworkState";
    public NetworkState(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        form.registerForOnDestroy(this);
        networkStateReceiver = new NetworkStateReceiver();
        registerNetworkStateMonitor();

        Log.d(LOG_TAG, "NetworkState Created" );
    }

    //实现广播接收器，接收和处理系统广播
    private class NetworkStateReceiver extends BroadcastReceiver {
        public NetworkStateReceiver() {

        }

        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if(ConnectivityManager.CONNECTIVITY_ACTION.equals(action)){
                networkState = getNetworkState(context);

                NetworkState(networkState);
            }
        }
    }

    //获取网络状态：WIFI、移动网络，还是未连接网络
    public String getNetworkState(Context context) {
        ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystem
```

```
            Service(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
            if (networkInfo != null) {
            if (networkInfo.isConnected()) {
                int type = networkInfo.getType();

                if (type == ConnectivityManager.TYPE_WIFI) {
                    return "WIFI";
                } else if (type == ConnectivityManager.TYPE_MOBILE) {
                    return getNetworkClass(context);
                }
            } else {
                return "未连接网络";
            }
        }

        return "未连接网络";
    }

    //当通过移动网络上网时，获取具体的网络类型：2G、3G、4G，还是未知网络
    public String getNetworkClass(Context context) {
        TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService
        (Context.TELEPHONY_SERVICE);

        switch (telephonyManager.getNetworkType()) {
            case TelephonyManager.NETWORK_TYPE_GPRS:
            case TelephonyManager.NETWORK_TYPE_EDGE:
            case TelephonyManager.NETWORK_TYPE_CDMA:
            case TelephonyManager.NETWORK_TYPE_1xRTT:
            case TelephonyManager.NETWORK_TYPE_IDEN:
            {
                return "2G";
            }

            case TelephonyManager.NETWORK_TYPE_UMTS:
            case TelephonyManager.NETWORK_TYPE_EVDO_0:
            case TelephonyManager.NETWORK_TYPE_EVDO_A:
            case TelephonyManager.NETWORK_TYPE_HSDPA:
            case TelephonyManager.NETWORK_TYPE_HSUPA:
            case TelephonyManager.NETWORK_TYPE_HSPA:
            case TelephonyManager.NETWORK_TYPE_EVDO_B:
            case TelephonyManager.NETWORK_TYPE_EHRPD:
            case TelephonyManager.NETWORK_TYPE_HSPAP:
            {
                return "3G";
```

```
                }

                case TelephonyManager.NETWORK_TYPE_LTE:
                {
                    return "4G";
                }

                default:
                    return "未知网络";
            }
        }

        //注册广播接收器
        private void registerNetworkStateMonitor(){
            IntentFilter intentFilter = new IntentFilter();
            intentFilter.addAction(ConnectivityManager.CONNECTIVITY_ACTION);
            context.registerReceiver(networkStateReceiver, intentFilter);
        }

        //注销广播接收器
        private void unregisterNetworkStateMonitor(){
            context.unregisterReceiver(networkStateReceiver);
        }

        //销毁当前屏幕时，注销广播接收器
        @Override
        public void onDestroy() {
            unregisterNetworkStateMonitor();
        }

        @SimpleEvent(description = "反馈网络状态，参数的值为以下字符串：WIFI、2G、3G、4G、未连接网络、未知网络")
        public void NetworkState(String state) {
            EventDispatcher.dispatchEvent(this, "NetworkState", state);
        }

        @SimpleFunction(description = "获取网络状态，返回值为以下字符串：WIFI、2G、3G、4G、未连接网络、未知网络")
        public String GetNetworkState() {
            return getNetworkState(context);
        }

    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如

图 19-17 所示。

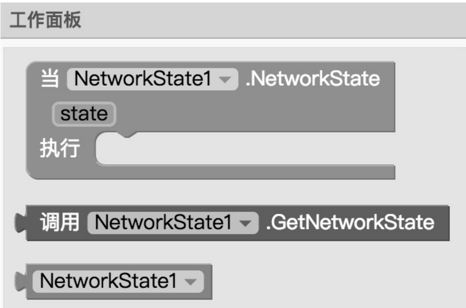


图 19-17　获取和监测设备网络状态插件函数

19.5.2　插件的使用

如图 19-18 所示，代码用于在通过网络下载文件时，根据不同的网络状态，给用户不同的提示。当网络处于 2G、3G 和 4G 状态时，提示用户是否暂停下载文件，当网络处于 WIFI 状态时，提示用户是否下载文件。

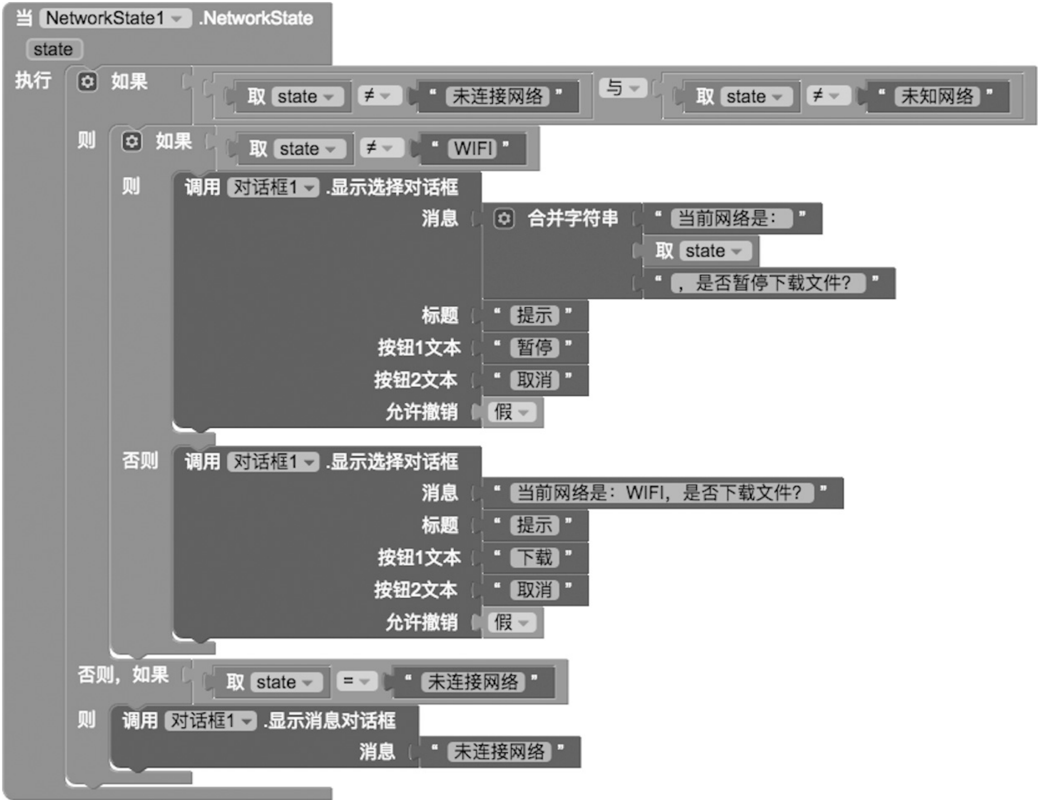


图 19-18　获取和监测设备网络状态插件使用代码 1

在浏览网页时，因为网页上往往有许多图片，这对网络状态要求也比较高，也应根据不同的网络状态，给用户不同的提示，代码如图 19-19 所示。

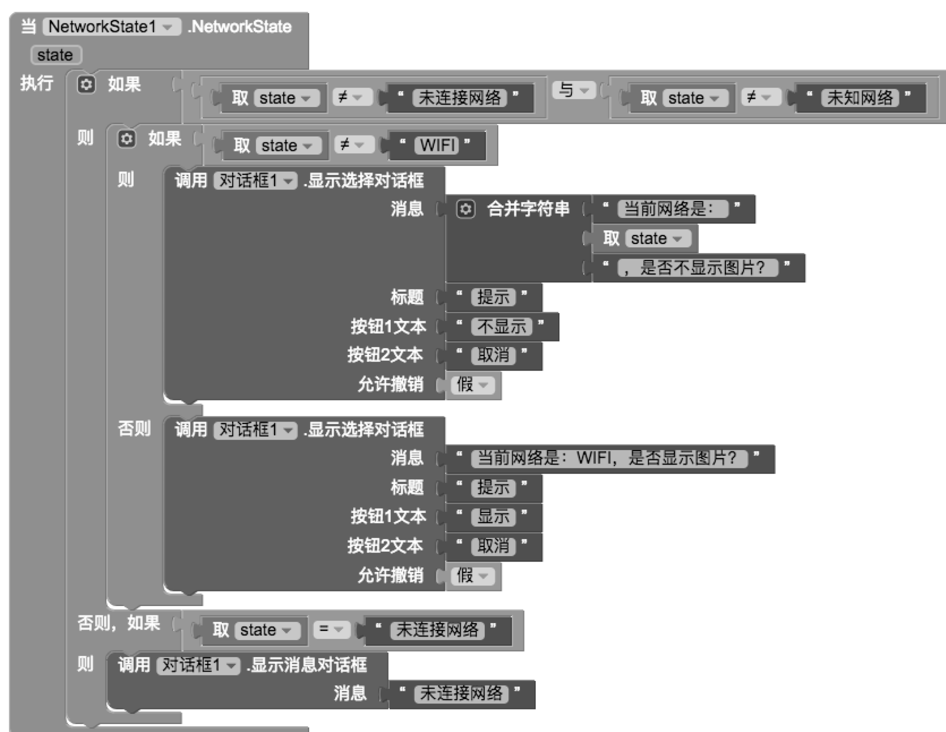


图 19-19　获取和监测设备网络状态插件使用代码 2

当网络处于 2G、3G 和 4G 状态时，提示用户是否不显示图片，当网络处于 WIFI 状态时，提示用户是否显示图片。

此插件也提供了接口函数，获取网络状态。每次发起网络请求时，可用此函数判断当前的网络状态，代码如图 19-20 所示。



图 19-20　获取和监测设备网络状态插件使用代码 3

19.6　获取和监测设备飞行模式状态插件

当移动设备处于飞行模式状态时，是不能上网和拨打电话的，Android 系统提供了系统广播，用于监测设备的飞行模式状态，也可以利用 Settings 类，获取飞行模式状态。具体的实现方式如下。

19.6.1 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 AirPlaneState.java，其中的代码如下：

```
package com.airplanestate.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.provider.Settings;

@DesignerComponent(version = AirPlaneState.VERSION,
    description = "功能：获取和监测飞行模式状态    开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class AirPlaneState extends AndroidNonvisibleComponent implements Component, OnDestroy
Listener {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;
    private final AirPlaneStateReceiver airPlaneStateReceiver;

    private static final String LOG_TAG = "AirPlaneState";
    public AirPlaneState(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        form.registerForOnDestroy(this);
        airPlaneStateReceiver = new AirPlaneStateReceiver();
```

```
        registerAirPlaneStateMonitor();

        Log.d(LOG_TAG, "AirPlaneState Created" );
    }

    private class AirPlaneStateReceiver extends BroadcastReceiver {
        public AirPlaneStateReceiver() {

        }

        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if(Intent.ACTION_AIRPLANE_MODE_CHANGED.equals(action)){
                boolean state = intent.getExtras().getBoolean("state");
                if (state){
                    AirPlaneState("开启");
                } else {
                    AirPlaneState("关闭");
                }
            }
        }
    }

    private void registerAirPlaneStateMonitor(){
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
        context.registerReceiver(airPlaneStateReceiver, intentFilter);
    }

    private void unregisterAirPlaneStateMonitor(){
        context.unregisterReceiver(airPlaneStateReceiver);
    }

    @Override
    public void onDestroy() {
        unregisterAirPlaneStateMonitor();
    }

    @SimpleEvent(description = "反馈飞行模式状态，参数值为开启、关闭")
    public void AirPlaneState(String state) {
        EventDispatcher.dispatchEvent(this, "AirPlaneState", state);
    }

    @SimpleFunction(description = "获取飞行模式状态，返回值为开启、关闭")
    public String GetAirPlaneState() {
```

```
                int state = Settings.System.getInt(context.getContentResolver(), Settings.System.
                AIRPLANE_ MODE_ON, 0);
                if (state == 1){
                    return "开启";
                } else {
                    return "关闭";
                }
            }

        }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-21 所示。



图 19-21　获取和监测飞行模式状态插件函数

19.6.2　插件的使用

使用示例代码如图 19-22 所示。



图 19-22　获取和监测飞行模式状态插件使用代码

19.7　系统设置插件

此插件用于从用户操作的当前界面跳转到系统设置界面，以方便用户进行各种设置。

19.7.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件

SystemSettings.java，其中的代码如下：

```
package com.systemsettings.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.Context;
import android.content.Intent;
import android.provider.Settings;

@DesignerComponent(version = SystemSettings.VERSION,
    description = "功能：跳转到系统设置界面    开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class SystemSettings extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;

    private static final String LOG_TAG = "SystemSettings";
    public SystemSettings(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "SystemSettings Created" );
    }

    @SimpleFunction(description = "跳转到系统设置界面")
    public void SystemSettings() {
        Intent intent = new Intent(Settings.ACTION_SETTINGS);
        context.startActivity(intent);
    }

    @SimpleFunction(description = "跳转到无线和网络设置界面")
    public void WirelessSettings() {
```

```
            Intent intent = new Intent(Settings.ACTION_WIRELESS_SETTINGS);
            context.startActivity(intent);
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-23 所示。

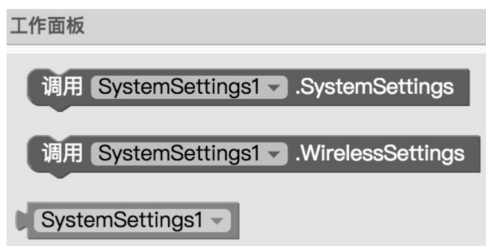


图 19-23　系统设置插件函数

19.7.2　插件的使用

在前面的例子中，当应用检测到设备没有连接网络和设备处于飞行模式状态时，只是给用户提示；结合此插件，可以从应用界面跳转到系统设置界面来核查网络设置或关闭飞行模式，用户体验更好。

具体代码如图 19-24～图 19-27 所示。



图 19-24　检测设备网络状态



图 19-25　系统设置插件使用代码 1

（用户单击“确定”按钮，则会跳转到无线和网络设置界面，以便用户核查网络设置。）



图 19-26　检测设备飞行模式状态



图 19-27　系统设置插件使用代码 2

（用户单击“确定”按钮，则会跳转到系统设置界面，以便用户关闭飞行模式。）

19.8　列表插件

用 App Inventor 2 开发应用的时候，列表是常使用的功能模块，但系统提供的列表相关功能不全面，创建列表的方式不够方便灵活，一些常用功能，如列表排序，在列表中查找最大值、最小值的功能也未提供。

现开发一个插件让开发人员能够比较灵活地创建列表，以及提供列表排序、查找最大值和最小值的功能，具体实现方式如下。

19.8.1　列表创建

使用以下函数，开发人员可以把字符串和分隔符作为创建列表的参数，这样，只需一行调用此函数的代码，就可创建列表，非常方便灵活。

```
@SimpleFunction(description = "把包含分隔符的字符串按分隔符分解，并转成列表")
public static YailList ListCreate(String itemString, String separator){
    YailList items = new YailList();
    if (itemString.length() > 0) {
        items = YailList.makeList((Object[]) itemString.split(separator));
```

```
            }

            return items;
        }
```

19.8.2 列表排序

Java 语言提供了 Collections 类对列表进行处理，功能强大，且使用非常方便。但 App Inventor 2 使用的列表类是 YailList，需要先把 YailList 列表转换成标准的 Java 列表后，再使用 Collections 类进行处理。

Collections 类对数字类型列表和非数字类型列表的处理方式不完全一样，如 8 和 79 两个数字排序，按数字类型处理 79 比 8 大，按非数字类型处理 8 比 79 大，这样代码也要分别实现。

排序函数的代码如下：

```
        @SimpleFunction(description = "列表升序排序")
        public YailList ListSort(YailList list, boolean isNumber) {
            if (!isNumber){
                Object[] objectsArray = list.toArray();
                java.util.List objectsList = Arrays.asList(objectsArray);
                java.util.Collections.sort(objectsList);

                return YailList.makeList(objectsList);
            } else {
                //int 和 float 类型都可以看成是 double 的子类，数字类型统一按 double 类型处理
                Object[] objectsArray = list.toArray();

                java.util.List <Double> doublesArray = new ArrayList<Double>();

                //把 Object 数组转成 double 数组
                for(int i = 0; i < objectsArray.length; i++){
                    doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
                }

                java.util.Collections.sort(doublesArray);

                return YailList.makeList(doublesArray);
            }
        }

        @SimpleFunction(description = "列表降序排序")
        public YailList ListReverse(YailList list, boolean isNumber) {
            if (!isNumber){
                Object[] objectsArray = list.toArray();
                java.util.List objectsList = Arrays.asList(objectsArray);
```

```
            java.util.Collections.sort(objectsList);
            java.util.Collections.reverse(objectsList);

            return YailList.makeList(objectsList);
        } else {
            Object[] objectsArray = list.toArray();

            java.util.List <Double> doublesArray = new ArrayList<Double>();

            for(int i = 0; i < objectsArray.length; i++){
                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
            }

            java.util.Collections.sort(doublesArray);
            java.util.Collections.reverse(doublesArray);

            return YailList.makeList(doublesArray);
        }
    }
```

19.8.3 列表查找

使用 Collections 类的 max 和 min 函数，在列表中查找最大值和最小值，对数字类型列表和非数字类型列表的处理方式也不一样。因为列表元素的类型不确定，所以函数的返回值类型为 Object。

查找函数的代码如下：

```
    @SimpleFunction(description = "列表最大值")
    public Object ListMax(YailList list, boolean isNumber) {
        if (!isNumber){
            Object[] objectsArray = list.toArray();
            java.util.List objectsList = Arrays.asList(objectsArray);
            return java.util.Collections.max(objectsList);
        } else {
            Object[] objectsArray = list.toArray();

            java.util.List <Double> doublesArray = new ArrayList<Double>();

            for(int i = 0; i < objectsArray.length; i++){
                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
            }

            return java.util.Collections.max(doublesArray);
        }
    }
```

```
    @SimpleFunction(description = "列表最小值")
    public Object ListMin(YailList list, boolean isNumber) {
        if (!isNumber){
            Object[] objectsArray = list.toArray();
            java.util.List objectsList = Arrays.asList(objectsArray);
            return java.util.Collections.min(objectsList);
        } else {
            Object[] objectsArray = list.toArray();

            java.util.List <Double> doublesArray = new ArrayList<Double>();

            for(int i = 0; i < objectsArray.length; i++){
                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
            }

            return java.util.Collections.min(doublesArray);
        }
    }
```

19.8.4　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 QZList.java，完整的代码如下：

```
package com.qzlist.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;
import com.google.appinventor.components.runtime.util.YailList;

import android.content.Context;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collection;

@DesignerComponent(version = QZList.VERSION,
    description = "功能：列表处理　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
```

```
        nonVisible = true,
        iconName = "images/extension.png")

@SimpleObject(external = true)
public class QZList extends AndroidNonvisibleComponent {
        public static final int VERSION = 1;
        private ComponentContainer container;
        private Context context;

        private static final String LOG_TAG = "QZList";
        public QZList(ComponentContainer container) {
                super(container.$form());
                this.container = container;
                context = (Context) container.$context();

                Log.d(LOG_TAG, "QZList Created" );
        }

        @SimpleFunction(description = "把包含分隔符的字符串按分隔符分解，并转成列表")
        public static YailList ListCreate(String itemString, String separator){
                YailList items = new YailList();
                if (itemString.length() > 0) {
                        items = YailList.makeList((Object[]) itemString.split(separator));
                }

                return items;
        }

        @SimpleFunction(description = "列表升序排序")
        public YailList ListSort(YailList list, boolean isNumber) {
                if (!isNumber){
                        Object[] objectsArray = list.toArray();
                        java.util.List objectsList = Arrays.asList(objectsArray);
                        java.util.Collections.sort(objectsList);

                        return YailList.makeList(objectsList);
                } else {
                        Object[] objectsArray = list.toArray();

                        java.util.List <Double> doublesArray = new ArrayList<Double>();

                        for(int i = 0; i < objectsArray.length; i++){
                                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
                        }
```

```
            java.util.Collections.sort(doublesArray);

            return YailList.makeList(doublesArray);
        }
    }

    @SimpleFunction(description = "列表降序排序")
    public YailList ListReverse(YailList list, boolean isNumber) {
        if (!isNumber){
            Object[] objectsArray = list.toArray();
            java.util.List objectsList = Arrays.asList(objectsArray);
            java.util.Collections.sort(objectsList);
            java.util.Collections.reverse(objectsList);

            return YailList.makeList(objectsList);
        } else {
            Object[] objectsArray = list.toArray();

            java.util.List <Double> doublesArray = new ArrayList<Double>();

            for(int i = 0; i < objectsArray.length; i++){
                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
            }

            java.util.Collections.sort(doublesArray);
            java.util.Collections.reverse(doublesArray);

            return YailList.makeList(doublesArray);
        }
    }

    @SimpleFunction(description = "列表最大值")
    public Object ListMax(YailList list, boolean isNumber) {
        if (!isNumber){
            Object[] objectsArray = list.toArray();
            java.util.List objectsList = Arrays.asList(objectsArray);
            return java.util.Collections.max(objectsList);
        } else {
            Object[] objectsArray = list.toArray();

            java.util.List <Double> doublesArray = new ArrayList<Double>();

            for(int i = 0; i < objectsArray.length; i++){
                doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
            }
```

```
                return java.util.Collections.max(doublesArray);
            }
        }

        @SimpleFunction(description = "列表最小值")
        public Object ListMin(YailList list, boolean isNumber) {
            if (!isNumber){
                Object[] objectsArray = list.toArray();
                java.util.List objectsList = Arrays.asList(objectsArray);
                return java.util.Collections.min(objectsList);
            } else {
                Object[] objectsArray = list.toArray();

                java.util.List <Double> doublesArray = new ArrayList<Double>();

                for(int i = 0; i < objectsArray.length; i++){
                    doublesArray.add(Double.parseDouble(objectsArray[i].toString()));
                }

                return java.util.Collections.min(doublesArray);
            }
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-28 和图 19-29 所示。



图 19-28　列表插件函数 1



图 19-29　列表插件函数 2

19.8.5　插件的使用

使用示例代码如图 19-30 所示。



图 19-30　列表插件使用代码

19.9　代数计算插件

App Inventor 2 的数学模块提供了许多功能，但未提供矩阵和复数计算功能，现开发一个插件提供矩阵和复数计算功能。具体实现如下。

19.9.1　矩阵的运算

矩阵运算实现了矩阵的相加和相减的功能。无论是几维矩阵，都可以表示为一维数组，也就是可以使用列表表示矩阵，在 App Inventor 2 中就是 YailList 列表。

数字的类型有三种：int、float 和 double，int 和 float 可以看成是 double 的子集；因不确定矩阵元素的类型，把矩阵元素的类型定为 double。

矩阵运算函数的代码如下：

```
@SimpleFunction(description = "矩阵相加")
public YailList MatrixAdd(YailList list1, YailList list2) {
    java.util.List<Double> resultsList = new ArrayList<Double>();

    Object[] objectsArray1 = list1.toArray();
    Object[] objectsArray2 = list2.toArray();

    if ((list1.size() == list2.size()) && (list1.size() > 0)){
        for (int i = 0; i < list1.size(); i++){
```

```
                resultsList.add(new Double(objectsArray1[i].toString()) + new Double(objects
                Array2[i].toString()));
            }

            return YailList.makeList(resultsList);
        } else {
            return null;
        }

    }

    @SimpleFunction(description = "矩阵相减")
    public YailList MatrixSub(YailList list1, YailList list2) {
        java.util.List<Double> resultsList = new ArrayList<Double>();

        Object[] objectsArray1 = list1.toArray();
        Object[] objectsArray2 = list2.toArray();

        if ((list1.size() == list2.size()) && (list1.size() > 0)){
            for (int i = 0; i < list1.size(); i++){
                resultsList.add(new Double(objectsArray1[i].toString()) - new Double (objects
                Array2[i].toString()));
            }

            return YailList.makeList(resultsList);
        } else {
            return null;
        }
    }
```

相加和相减函数的参数类型都是 YailList，返回值类型也是 YailList。

19.9.2　复数的运算

复数的运算实现了复数的加、减、乘、除和求模的功能。因不确定复数实部和虚部数字的具体类型，就把类型都定为 double。

复数的运算结果包含实部和虚部两个变量的数值，把运算结果中的实部和虚部作为事件响应函数的参数，利用事件响应函数返回计算结果。

复数运算函数的代码如下：

```
    @SimpleFunction(description = "复数运算：+、-、x、/")
    // operator 的值可以是"+"、"-"、"x"和"/"四个字符，对应加减乘除运算
    public void Complex(double real1, double imag1, double real2, double imag2, String operator) {
        if (operator.equals("+")){
            complexAdd(real1, imag1, real2, imag2);
        }
```

```
        if (operator.equals("-")){
            complexSub(real1, imag1, real2, imag2);
        }

        if (operator.equals("x")){
            complexMul(real1, imag1, real2, imag2);
        }

        if (operator.equals("/")){
            complexDiv(real1, imag1, real2, imag2);
        }

        ComplexResult(real, imag);
    }

    @SimpleFunction(description = "复数的模")
    public double ComplexModel(double real, double imag) {
        return Math.sqrt(real * real + imag * imag);
    }

    @SimpleEvent(description = "反馈计算结果")
    public void ComplexResult(double real, double imag) {
        EventDispatcher.dispatchEvent(this, "ComplexResult", real, imag);
    }
```

19.9.3 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 AlgebraicOperation.java，代码如下：

```
package com.algebraicoperation.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;
import com.google.appinventor.components.runtime.util.YailList;

import android.content.Context;
import java.util.Arrays;
import java.util.ArrayList;
```

```
import java.lang.String;

@DesignerComponent(version = AlgebraicOperation.VERSION,
    description = "功能：代数运算    开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class AlgebraicOperation extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;
    private double real;
    private double imag;

    private static final String LOG_TAG = "AlgebraicOperation";
    public AlgebraicOperation(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "AlgebraicOperation Created" );
    }

    @SimpleFunction(description = "矩阵相加")
    public YailList MatrixAdd(YailList list1, YailList list2) {
        java.util.List<Double> resultsList = new ArrayList<Double>();

        Object[] objectsArray1 = list1.toArray();
        Object[] objectsArray2 = list2.toArray();

        if ((list1.size() == list2.size()) && (list1.size() > 0)){
            for (int i = 0; i < list1.size(); i++){
                resultsList.add(new Double(objectsArray1[i].toString()) + new Double(objectsArray2[i].
                toString()));
            }

            return YailList.makeList(resultsList);
        } else {
            return null;
        }

    }
```

```
@SimpleFunction(description = "矩阵相减")
public YailList MatrixSub(YailList list1, YailList list2) {
    java.util.List<Double> resultsList = new ArrayList<Double>();

    Object[] objectsArray1 = list1.toArray();
    Object[] objectsArray2 = list2.toArray();

    if ((list1.size() == list2.size()) && (list1.size() > 0)){
        for (int i = 0; i < list1.size(); i++){
            resultsList.add(new Double(objectsArray1[i].toString()) - new Double(objectsArray2[i].
            toString()));
        }

        return YailList.makeList(resultsList);
    } else {
        return null;
    }
}

@SimpleFunction(description = "复数运算：+、-、x、/")
public void Complex(double real1, double imag1, double real2, double imag2, String operator) {
    if (operator.equals("+")){
        complexAdd(real1, imag1, real2, imag2);
    }

    if (operator.equals("-")){
        complexSub(real1, imag1, real2, imag2);
    }

    if (operator.equals("x")){
        complexMul(real1, imag1, real2, imag2);
    }

    if (operator.equals("/")){
        complexDiv(real1, imag1, real2, imag2);
    }

    ComplexResult(real, imag);
}

@SimpleFunction(description = "复数的模")
public double ComplexModel(double real, double imag) {
    return Math.sqrt(real * real + imag * imag);
}
```

```
        @SimpleEvent(description = "反馈计算结果")
        public void ComplexResult(double real, double imag) {
            EventDispatcher.dispatchEvent(this, "ComplexResult", real, imag);
        }

        public void complexAdd(double real1, double imag1, double real2, double imag2) {
            real = real1 + real2;
            imag = imag1 + imag2;
        }

        public void complexSub(double real1, double imag1, double real2, double imag2) {
            real = real1 - real2;
            imag = imag1 - imag2;
        }

        public void complexMul(double real1, double imag1, double real2, double imag2) {
            real = real1 * real2 - imag1 * imag2;
            imag = imag1 * real2 + real1 * imag2;
        }

        public void complexDiv(double real1, double imag1, double real2, double imag2) {
            real = (real1 * real2 + imag * imag2) / (real2 * real2 + imag2 * imag2);
            imag = (imag1 * real2 - real1 * imag2) / (real2 * real2 + imag2 * imag2);
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-31 和图 19-32 所示。

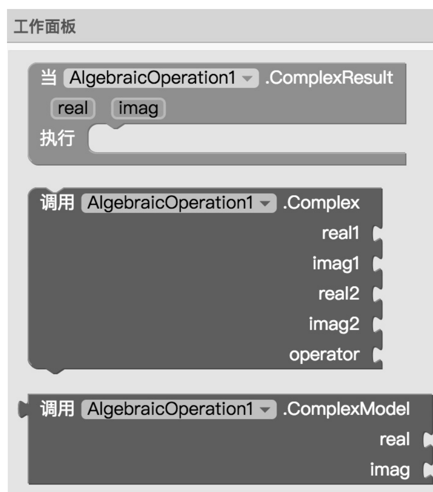


图 19-31　代数计算插件函数 1

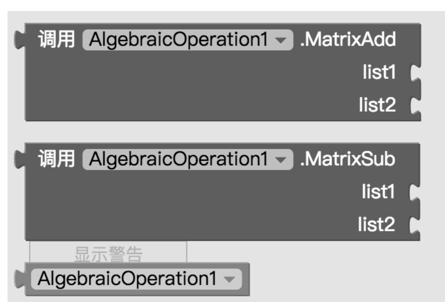


图 19-32　代数计算插件函数 2

19.9.4 插件的使用

两个 2×2 的矩阵相加减的代码如图 19-33 和图 19-34 所示。



图 19-33 代数计算插件使用代码 1

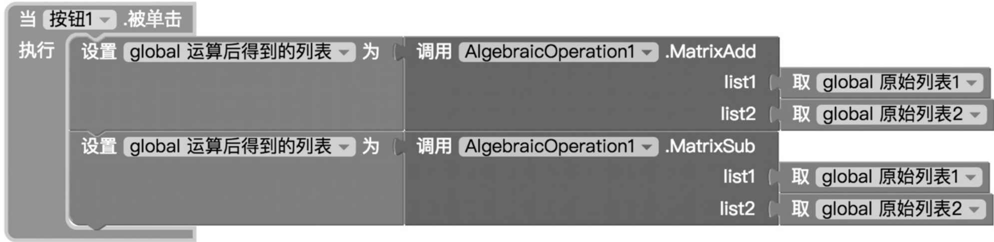


图 19-34 代数计算插件使用代码 2

复数运算的代码如图 19-35 和图 19-36 所示。



图 19-35 代数计算插件使用代码 3



图 19-36 代数计算插件使用代码 4

19.10 几何计算插件

App Inventor 2 的数学模块未提供计算图形面积和体积的功能，现开发一个插件提供这两种功能。具体实现如下。

19.10.1 计算图形的面积

长方形和正方形统一使用矩形表示，面积计算方式：宽度×高度。

三角形的面积有多种计算方式，此插件实现了两种：

① 面积=底×高/2；

② 面积=边长 1×边长 2×两边间的夹角的正弦值/2。

长方体和正方体统一使用立方体表示，表面积计算方式：长度×宽度×2+长度×高度×2+宽度×高度×2。

19.10.2 计算图形的体积

长方体和正方体统一使用立方体表示，体积计算方式：长度×宽度×高度。

19.10.3 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 GeometricOperation.java，代码如下：

```
package com.geometricoperation.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;
import com.google.appinventor.components.runtime.util.YailList;

import android.content.Context;
```

```
import java.util.Arrays;
import java.util.ArrayList;
import java.lang.String;

@DesignerComponent(version = GeometricOperation.VERSION,
    description = "功能：几何运算　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class GeometricOperation extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;

    private static final String LOG_TAG = "GeometricOperation";
    public GeometricOperation(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG , "GeometricOperation Created" );
    }

    @SimpleFunction(description = "矩形面积")
    public double RectangularArea(double width, double height) {
        return (width * height);
    }

    @SimpleFunction(description = "三角形面积")
    public double TriangleArea(double bottom, double height) {
        return (bottom * height / 2);
    }

    @SimpleFunction(description = "三角形面积")
    public double TriangleArea2(double sideLength1, double sideLength2, double angle) {
        return (sideLength1 * sideLength2 * Math.sin(angle)/ 2);
    }

    @SimpleFunction(description = "圆形面积")
    public double CircularArea(double radius, double pi) {
        return (pi * radius * radius);
```

```
        }

        @SimpleFunction(description = "椭圆面积")
        public double EllipticalArea(double longAxis, double shortAxis, double pi) {
            return (pi * longAxis * shortAxis / 4);
        }

        @SimpleFunction(description = "梯形面积")
        public double TrapezoidalArea(double top, double bottom, double height) {
            return ((top + bottom) * height / 2);
        }

        @SimpleFunction(description = "立方体表面积")
        public double CubicArea(double length, double width, double height) {
            return (length * width * 2 + length * height * 2 + width * height * 2);
        }

        @SimpleFunction(description = "立方体体积")
        public double CubicVolume(double length, double width, double height) {
            return (length * width * height);
        }

        @SimpleFunction(description = "圆柱体表面积")
        public double CylinderArea(double radius, double height, double pi) {
            return (pi * radius * radius * 2 + pi * radius * 2 * height);
        }

        @SimpleFunction(description = "圆柱体体积")
        public double CylinderVolume(double radius, double height, double pi) {
            return (pi * radius * radius * height);
        }

        @SimpleFunction(description = "球体表面积")
        public double SphereArea(double radius, double pi) {
            return (pi * radius * radius * 4);
        }

        @SimpleFunction(description = "球体体积")
        public double SphereVolume(double radius, double pi) {
            return ((pi * radius * radius * radius * 4) / 3);
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-37～图 19-40 所示。

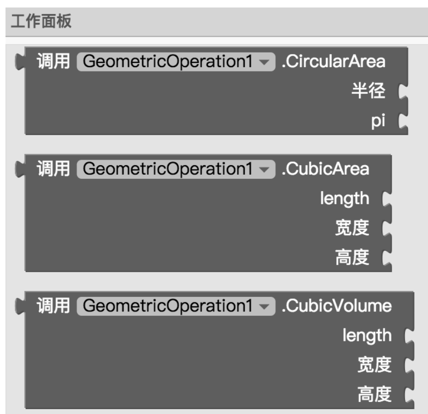


图 19-37　几何计算插件函数 1



图 19-38　几何计算插件函数 2

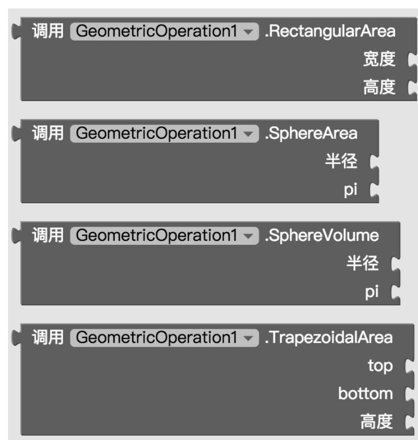


图 19-39　几何计算插件函数 3

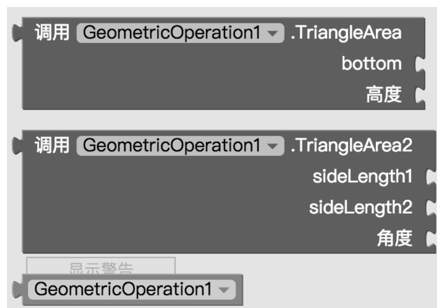


图 19-40　几何计算插件函数 4

19.11　手电筒功能插件

此插件是利用移动设备上的闪光灯实现手电筒功能。具体实现如下。

19.11.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 Torch.java，其中的代码如下：

```
package com.torch.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
```

```
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.Context;
import android.content.Intent;
import android.hardware.Camera;
import android.hardware.Camera.Parameters;

@DesignerComponent(version = Torch.VERSION,
    description = "功能：手电筒　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
//申请使用 CAMERA 和闪光灯的系统权限
@UsesPermissions(permissionNames = "android.permission.CAMERA, android.permission. FLASHLIGHT")
public class Torch extends AndroidNonvisibleComponent implements OnDestroyListener {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;

    private Camera camera;

    private static final String LOG_TAG = "Torch";
    public Torch(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "Torch Created" );
    }

   //销毁当前界面时，关闭 CAMERA
    @Override
    public void onDestroy() {
        if(camera != null){
            camera.stopPreview();
            camera.release();
        }
    }

    @SimpleFunction(description = "打开手电筒")
    public void TorchOn() {
```

```
            camera = Camera.open();

            Parameters params = camera.getParameters();
            params.setFlashMode(Parameters.FLASH_MODE_TORCH);

            camera.setParameters(params);
            camera.startPreview();
        }

        @SimpleFunction(description = "关闭手电筒")
        public void TorchOff() {
            if(camera != null){
                Parameters params = camera.getParameters();
                params.setFlashMode(Parameters.FLASH_MODE_OFF);

                camera.setParameters(params);
                camera.startPreview();
            }
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-41 所示。



图 19-41　手电筒插件函数

19.11.2　插件的使用

示例代码如图 19-42 所示。



图 19-42　手电筒插件使用代码

单击按钮打开和关闭手电筒的时候，按钮上的文字在“打开”和“关闭”间切换。

19.12　手机号码校验插件

App Inventor 2 提供了发短信和拨打电话的功能，但没有提供对电话号码进行验证的功能；目前大多数应用在发短信和拨打电话号码的时候，都会对手机号码进行验证，以应对用户输错手机号码之类的情况。现开发一个插件提供上述功能，具体实现方式如下。

19.12.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 MobilePhoneNumber.java，其中的代码如下：

```
package com.mobilephonenumber.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import android.content.Context;

@DesignerComponent(version = MobilePhoneNumber.VERSION,
    description = "功能：验证手机号码格式　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class MobilePhoneNumber extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;

    private static final String LOG_TAG = "MobilePhoneNumber";
    public MobilePhoneNumber(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "MobilePhoneNumber Created" );
```

```
        }

        @SimpleFunction(description = "验证手机号码格式是否正确，返回值为 true 或 false")
        public boolean IsMobilePhoneNumber(String phoneNumber) {
            //用于校验手机号码的正则表达式
            String phoneNumberRegularExpression = "[1][3578]\\d{9}";

            if (phoneNumber == null || phoneNumber.length() != 11)
            {
                return false;
            }

            return phoneNumber.matches(phoneNumberRegularExpression);
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-43 所示。

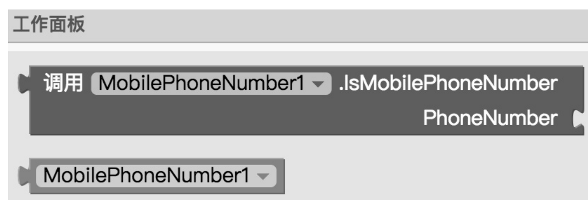


图 19-43　手机号码校验插件函数

19.12.2　插件的使用

示例代码如图 19-44 所示。

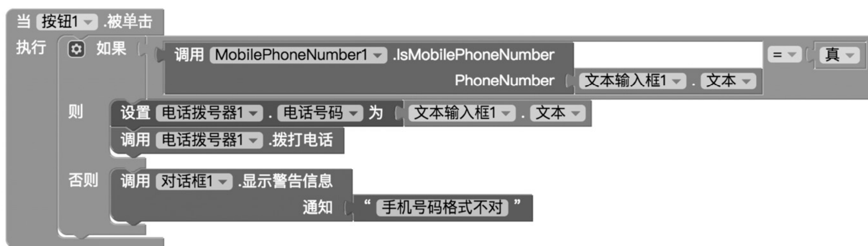


图 19-44　手机号码校验插件使用代码

19.13　判断应用是否第一次运行插件

许多应用在用户第一次使用的时候，会显示欢迎界面、使用说明之类的信息和设置某些参数的值，这就需要判断应用是否是第一次运行。

在 App Inventor 2 中，可以参照微数据库组件，利用 Android 系统的 SharedPreferences 框架，把标识应用是否第一次运行的变量值保存在 xml 文件中，通过读取 xml 文件中的数值判断应用是否第一次运行。

具体实现方式如下。

19.13.1 插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 FirstRun.java，其中的代码如下：

```
package com.firstrun.extensions;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.*;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;
import com.google.appinventor.components.runtime.util.*;

import com.google.appinventor.components.runtime.errors.YailRuntimeError;
import android.content.SharedPreferences;
import org.json.JSONException;

@DesignerComponent(version = FirstRun.VERSION,
    description = "功能：判断 App 是否第一次运行    开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class FirstRun extends AndroidNonvisibleComponent implements Component {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;
    private SharedPreferences sharedPreferences;
    private final String tag = "FirstRun";
    private static final String LOG_TAG = "FirstRun";

    public FirstRun(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();
        Log.d(LOG_TAG, "FirstRun Created" );
        sharedPreferences = context.getSharedPreferences("FirstRun", Context.MODE_PRIVATE);
```

```
    }

    @SimpleFunction(description = "判断 App 是否第一次运行，返回值为 true 或 false")
    //从 xml 文件中读取的字符串值为 yes，则 App 是第一次运行，函数返回值为 true，否则返回
    //值为 false
    public boolean IsFirstRun() {
        if (GetValue(tag, "yes") == "yes") {
            StoreValue(tag, "no");
            return true;
        }else{
            return false;
        }
    }

    @SimpleFunction(description = "清除记录是否第一次运行的标识")
    public void ClearFirstRunFlag() {
        final SharedPreferences.Editor sharedPrefsEditor = sharedPreferences.edit();
        sharedPrefsEditor.clear();
        sharedPrefsEditor.commit();
    }

    //从 xml 文件中读取数值
    public Object GetValue(final String tag, final Object valueIfTagNotThere) {
        try {
            String value = sharedPreferences.getString(tag, "");
            return (value.length() == 0) ? valueIfTagNotThere : JsonUtil.getObjectFromJson(value);
        } catch (JSONException e) {
            throw new YailRuntimeError("Value failed to convert from JSON.", "JSON Creation Error.");
        }
    }

    //保存数值到 xml 文件中
    public void StoreValue(final String tag, final Object valueToStore) {
        final SharedPreferences.Editor sharedPrefsEditor = sharedPreferences.edit();
        try {
            sharedPrefsEditor.putString(tag, JsonUtil.getJsonRepresentation(valueToStore));
            sharedPrefsEditor.commit();
        } catch (JSONException e) {
            throw new YailRuntimeError("Value failed to convert to JSON.", "JSON Creation Error.");
        }
    }
}
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-45 所示。



图 19-45　判断应用是否第一次运行插件函数

19.13.2　插件的使用

示例代码如图 19-46 所示。



图 19-46　判断应用是否第一次运行插件使用代码

如果应用是第一次运行，就显示一张“欢迎.jpg”图片。

19.14　获取定位功能状态插件

定位功能属于常用功能，App Inventor 2 也提供了“位置传感器”组件，但需要用户打开设备的定位功能，“位置传感器”组件才能正常使用。在使用“位置传感器”组件的时候，最好先获取设备当前的定位功能状态，如果定位功能没有打开，则提示用户是否打开定位功能。App Inventor 2 没有提供获取定位功能状态的组件，需要自行开发，具体实现如下。

19.14.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 LocationState.java，其中的代码如下：

```
package com.locationstate.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;
```

```
import android.location.LocationManager;

@DesignerComponent(version = LocationState.VERSION,
    description = "功能：获取定位功能状态　开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
@UsesPermissions(permissionNames = "android.permission.ACCESS_FINE_LOCATION")
public class LocationState extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;

    private static final String LOG_TAG = "LocationState";
    public LocationState(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "LocationState Created" );
    }

    @SimpleFunction(description = "获取定位功能状态，返回值为开启或关闭")
    public String GetLocationState() {
        LocationManager locationManager = (LocationManager) context.getSystemService
        (Context.LOCATION_SERVICE);
        // 通过 GPS 卫星定位，适用于在室外定位
        boolean gpsProvider = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
        // 通过基站或 WIFI 定位，室内室外都可用
        boolean networkProvider = locationManager.isProviderEnabled(LocationManager.NETWORK_
        PROVIDER);
        if (gpsProvider || networkProvider) {
            return "开启";
        }

        return "关闭";
    }
}
```

“位置传感器”组件支持 GPS、基站或 WIFI 定位，所以只要用户打开其中任意一种定位功能，“位置传感器”组件都可正常使用。

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如

图 19-47 所示。



图 19-47　获取定位功能状态插件函数

19.14.2　插件的使用

示例代码如图 19-48 所示。



图 19-48　获取定位功能状态插件使用代码

在启动“位置传感器”组件之前，调用此插件获取定位功能状态，如果没有开启，则显示选择提示框，提示用户是否开启定位功能；如果用户单击确定按钮，则调用系统设置插件，跳转到系统设置界面，以便用户开启定位功能。

19.15　创建 JSON 字符串插件

目前应用和服务器间传输数据大多采用 JSON 格式的数据，App Inventor 2 提供了解析 JSON 格式数据的功能函数，但没有提供创建 JSON 格式数据的功能函数，需要自行开发创建 JSON 格式数据的插件，具体实现如下。

19.15.1　插件的实现

在存放插件源码的文件夹/appinventor/components/src/com/qz/extensions 下创建源码文件 JSONCreate.java，其中的代码如下：

```
package com.jsoncreate.extension;

import android.content.Context;
import android.util.Log;
import com.google.appinventor.components.annotations.DesignerComponent;
```

```
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.runtime.*;

import org.json.JSONException;
import org.json.JSONObject;

@DesignerComponent(version = JSONCreate.VERSION,
    description = "功能：创建 JSON 字符串    开发者：QZ",
    helpUrl = "http://blog.csdn.net/xjbclz",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    iconName = "images/extension.png")

@SimpleObject(external = true)
public class JSONCreate extends AndroidNonvisibleComponent {
    public static final int VERSION = 1;
    private ComponentContainer container;
    private Context context;
    private JSONObject obj;

    private static final String LOG_TAG = "JSONCreate";
    public JSONCreate(ComponentContainer container) {
        super(container.$form());
        this.container = container;
        context = (Context) container.$context();

        Log.d(LOG_TAG, "JSONCreate Created" );
    }

    @SimpleFunction(description = "创建 JSON 对象")
    public void CreateJSONObject() {
        obj = new JSONObject();
    }

    @SimpleFunction(description = "添加 boolean 类型元素")
    public void PutBooleanValue(String name, boolean booleanValue) {
        try {
            obj.put(name, booleanValue);
        }catch (JSONException e)
        {
            e.printStackTrace();
        }
```

```
    }

    @SimpleFunction(description = "添加 double 类型元素")
    public void PutDoubleValue(String name, double doubleValue) {
        try {
            obj.put(name, doubleValue);
        }catch (JSONException e)
        {
            e.printStackTrace();
        }
    }

    @SimpleFunction(description = "添加 int 类型元素")
    public void PutIntValue(String name, int intValue) {
        try {
            obj.put(name, intValue);
        }catch (JSONException e)
        {
            e.printStackTrace();
        }
    }

    @SimpleFunction(description = "添加 long 类型元素")
    public void PutLongValue(String name, long longValue) {
        try {
            obj.put(name, longValue);
        }catch (JSONException e)
        {
            e.printStackTrace();
        }
    }

    @SimpleFunction(description = "添加 Object 类型元素")
    public void PutObjectValue(String name, Object objectValue) {
        try {
            obj.put(name, objectValue);
        }catch (JSONException e)
        {
            e.printStackTrace();
        }
    }

    @SimpleFunction(description = "移除元素")
    public void Remove(String name) {
        obj.remove(name);
    }
```

```
        @SimpleFunction(description = "把 JSON 格式数据转成 String 类型数据")
        public String JSONtoString() {
            return String.valueOf(obj);
        }
    }
```

把插件导入项目，并添加到组件列表后，在工作面板中可以看到插件提供的函数，如图 19-49 和图 19-50 所示。

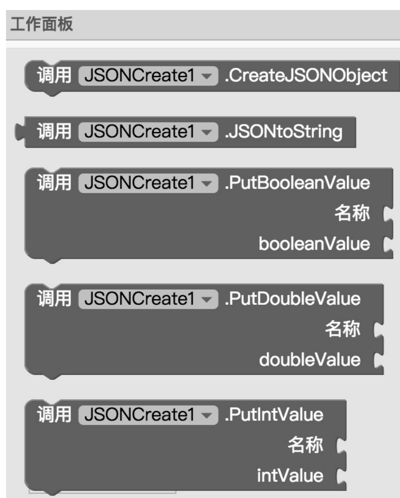


图 19-49　创建 JSON 字符串插件函数 1



图 19-50　创建 JSON 字符串插件函数 2

19.15.2　插件的使用

示例代码如图 19-51 所示。



图 19-51　创建 JSON 字符串插件使用代码

上述代码创建并向服务器发送如下 JSON 字符串：

```
{
    "man": true,
    "height": 175,
    "weight": 72.5,
    "name": "张三"
}
```

第 20 章　综合实例开发

20.1　抓青蛙游戏

20.1.1　游戏简介

抓青蛙游戏的玩法：有只青蛙在草地上跳来跳去，当玩家用手指单击青蛙时，就抓住了青蛙，同时积分加 1，每局的时间为 30s。

游戏结束时，如果积分达到或超过了 20 分，会在界面上显示一个点赞的图标，给玩家点赞。

玩家可以设置开启或关闭游戏的背景音乐，默认开启；也可以查看自己的积分记录。

20.1.2　游戏使用的素材

使用的素材如图 20−1 所示。

素材
caodi.png
dianzan.png
music.mid
qingwa.png

图 20−1　抓青蛙游戏使用的素材

caodi.png ——草地图片；
dianzan.png ——点赞图片；
music.mid ——背景音乐；
qingwa.png ——青蛙图片，也用做应用在系统桌面显示的 Icon 图片。

20.1.3　游戏使用的插件

使用的插件如图 20−2 所示。
BatteryState ——监测设备电量状态插件；
AirPlaneState ——获取和监测设备飞行模式状态插件；
QZList ——列表插件；
FirstRun ——判断应用是否第一次运行插件；
NetworkState ——获取和监测设备网络状态插件；

SystemInfo——获取设备和系统信息插件；

SystemSettings——系统设置插件。

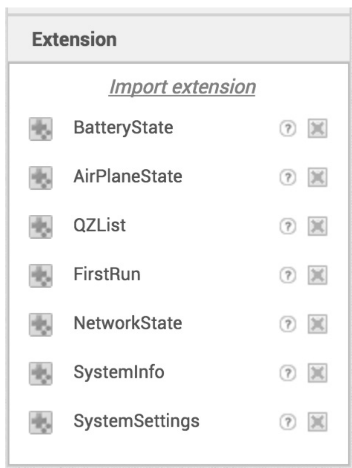


图 20-2　抓青蛙游戏使用的插件

20.1.4　游戏实现

1. 屏幕界面

此游戏有三个屏幕：主屏幕（Screen1）、设置背景音的屏幕（Screen2）和查看积分记录的屏幕（Screen3）。

图 20-3 为游戏未开始时的主屏幕界面，图 20-4 是游戏进行中的主屏幕界面。

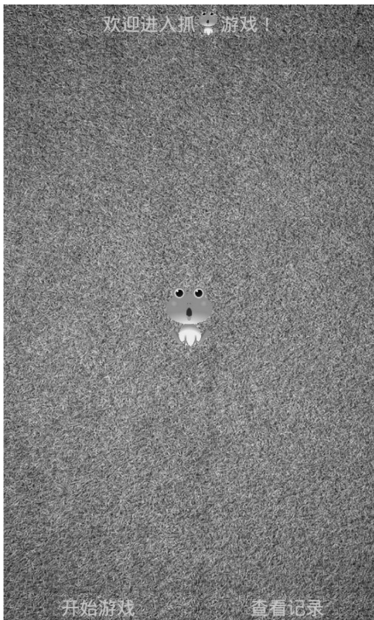


图 20-3　抓青蛙游戏主屏幕界面 1

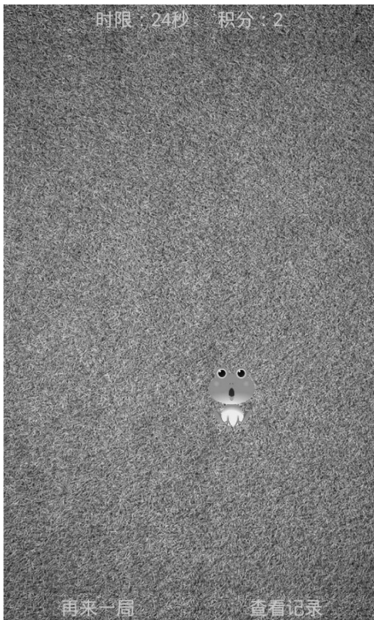


图 20-4　抓青蛙游戏主屏幕界面 2

图 20-5 是游戏结束时，积分小于 20 时的主屏幕界面，图 20-6 是积分大于 20 时显示的主屏幕界面。

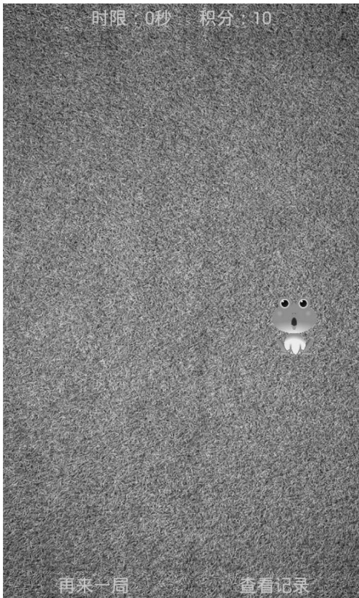


图 20-5　抓青蛙游戏主屏幕界面 3

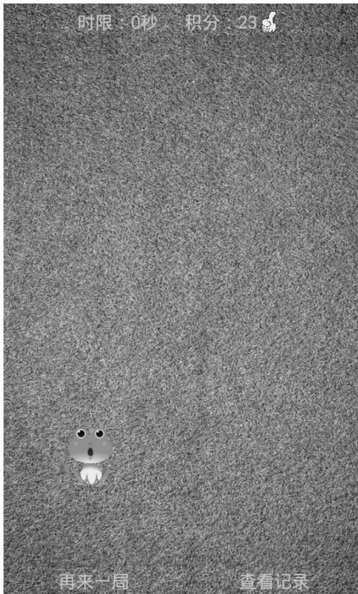


图 20-6　抓青蛙游戏主屏幕界面 4

主屏幕有“退出”“关于”“设置”和“更新版本”四个功能菜单，如图 20-7 所示。图 20-8 是单击“退出”菜单显示的界面，图 20-9 是单击“关于”菜单显示的界面。

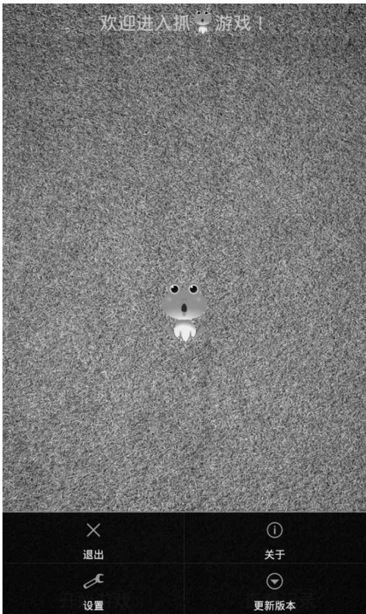


图 20-7　抓青蛙游戏主屏幕功能菜单



图 20-8　单击“退出”菜单显示的界面



图 20-9　单击“关于”菜单显示的界面

单击“设置”菜单，会打开设置背景音的屏幕；单击“更新版本”菜单，会打开一个网页。

图 20-10 为设置背景音的屏幕界面，图 20-11 为查看积分记录的屏幕界面，这两个屏幕都有“退出”和“关于”两个功能菜单。

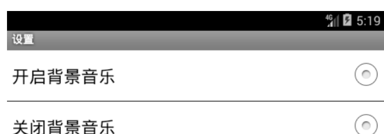




图 20-10　设置背景音的屏幕界面

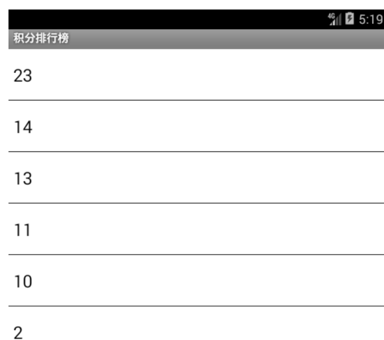




图 20-11　查看积分记录的屏幕界面

2. 主屏幕界面及功能实现

1）界面布局如图 20-12 所示。

2）界面使用的组件列表，如图 20-13 所示。

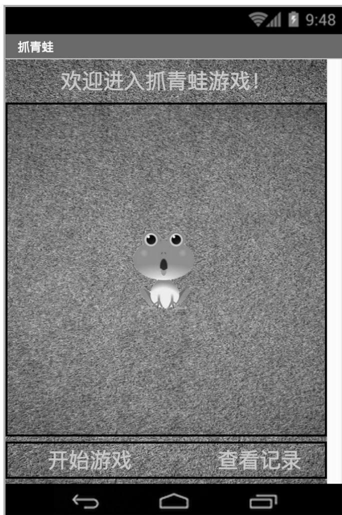


图 20-12　抓青蛙游戏主屏幕界面布局



图 20-13　抓青蛙游戏主屏幕的组件列表 1

“草地”画布中包含一个“青蛙”图像精灵，“功能标签水平布局”中包含“开始继续游戏”和“查看记录”两个标签，如图 20-14 所示。



图 20-14　抓青蛙游戏主屏幕的组件列表 2

各组件的属性设置见表 20-1。

表 20-1　抓青蛙游戏主屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen1	应用名称	抓青蛙
	背景图片	caodi.png
	图标	qingwa.png
	是否显示设置菜单	是
	是否显示更新版本菜单	是
	状态栏显示	否
	窗口大小	自动调整
	标题展示	否
数据显示	字号	24.0
	宽度	充满
	文本	欢迎进入抓青蛙游戏！
	文本对齐	居中
	文本颜色	青色
草地	背景图片	caodi.png
	高度	充满
	宽度	充满
青蛙	高度	80 像素
	宽度	80 像素
	图片	qingwa.png
	X 坐标	114
	Y 坐标	124
功能标签水平布局	背景颜色	透明
	宽度	充满
开始继续游戏	字号	24.0
	宽度	50 比例
	文本对齐	居中
	文本颜色	青色
查看记录	字号	24.0
	宽度	50 比例
	文本对齐	居中
	文本颜色	青色
游戏时间计时器	一直计时	否
	启用计时	否
音频播放器 1	循环播放	是
	源文件	music.mid

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

① 使用到的变量和初始值见表 20-2。

表 20-2　抓青蛙游戏主屏幕的全局变量

变 量 名 称	初　始　值
游戏积分	0
游戏剩余时间	0
游戏积分字符串	“”
播放背景音乐	“on”

② 屏幕初始化时运行的代码如图 20-15 所示。

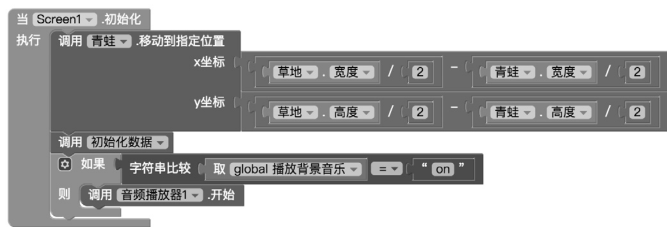


图 20-15　抓青蛙游戏主屏幕初始化时运行的代码

调用“青蛙”移动到指定位置函数移动“青蛙”图像精灵到“草地”画布中间。

“初始化数据”过程的具体代码如图 20-16 所示。

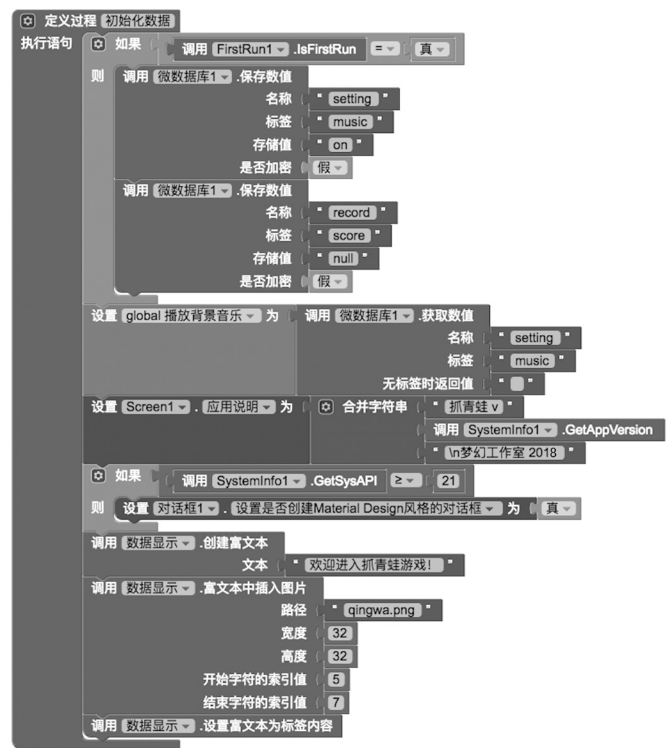


图 20-16　“初始化数据”过程代码

第一次运行应用时，设置播放背景音乐，对应的标签“music”的值为“on”，游戏的积分对应的标签“score”的值为“null”。

设置“Screen1”屏幕的“应用说明”内容为“抓青蛙+应用版本号+梦幻工作室 2018”，也就是单击“关于”菜单显示的内容。

Android 系统从 API 级别 21 开始才支持 Material Design 风格设计，调用“SystemInfo1”插件，判断当前系统的 API 级别是否大于等于 21，如果是则设置“对话框 1”的显示风格为 Material Design 风格。

为了丰富显示效果，利用富文本功能，把“数据显示”标签的文本内容“欢迎进入抓青蛙游戏!”中的“青蛙”两个字，用“qingwa.png”图片代替，如图 20-17 所示。



图 20-17　欢迎信息

③“开始继续游戏”标签的功能代码如图 20-18 所示。

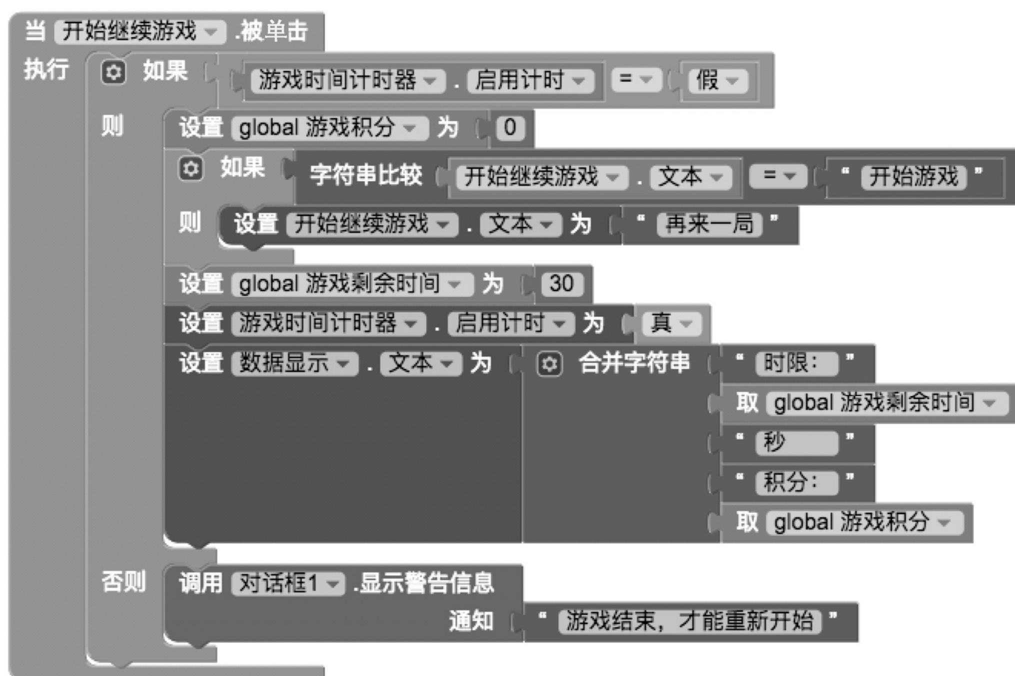


图 20-18 “开始继续游戏”标签的功能代码

如果标签的文本内容为“开始游戏”，说明用户正在玩第一局，文本内容改为“再来一局”。

设置全局变量“游戏剩余时间”的值为 30，也就是每局的时间为 30s。

④“游戏时间计时器”的功能代码如图 20-19 所示。

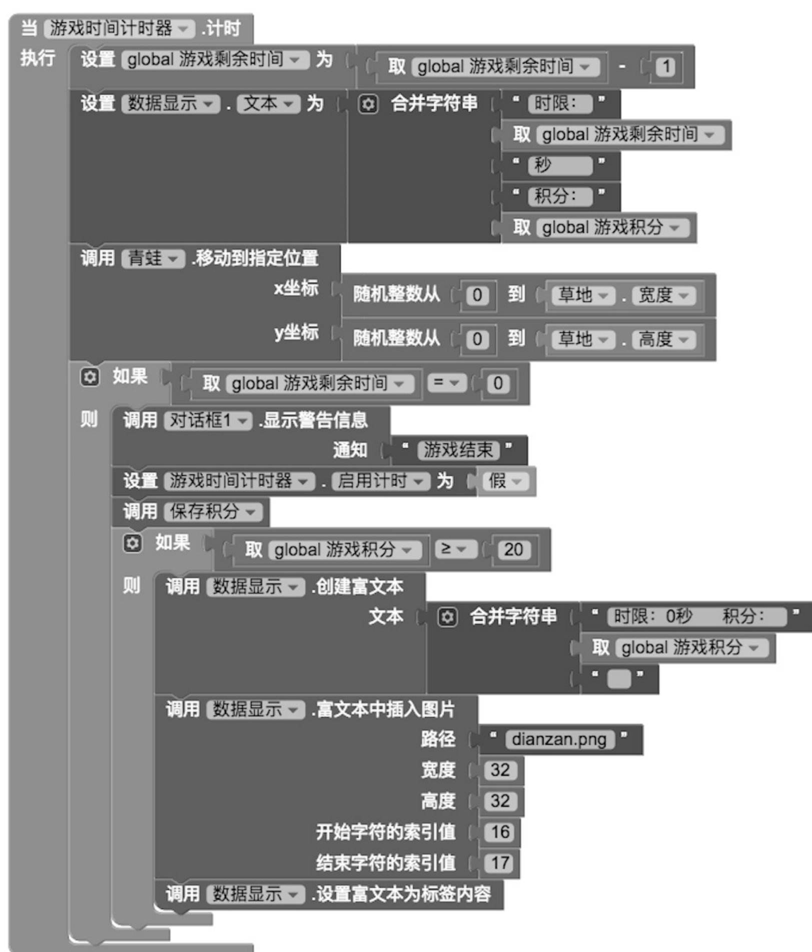


图 20-19 “游戏时间计时器”的功能代码

计时器的间隔时间为系统默认值:1s。

如果“游戏积分”的值大于等于 20，则利用富文本功能，在“数据显示”标签的文本内容最后显示一个“dianzan.png”图片，如图 20-20 所示。



图 20-20 积分提示

“保存积分”过程的代码如图 20-21 所示。

保存积分的时候，首先从“微数据库 1”里读取数值，并赋值给全局变量“游戏积分字符串”;

如果值为“null”，则还没有保存过积分，直接保存全局变量“游戏积分”的值到数据库里;

如果值不为“null”，则意味之前已经保存过积分，把“游戏积分字符串”的内容后面加

上逗号，再和这次的积分合并成一个字符串保存在数据库里，如图 20-22 所示。

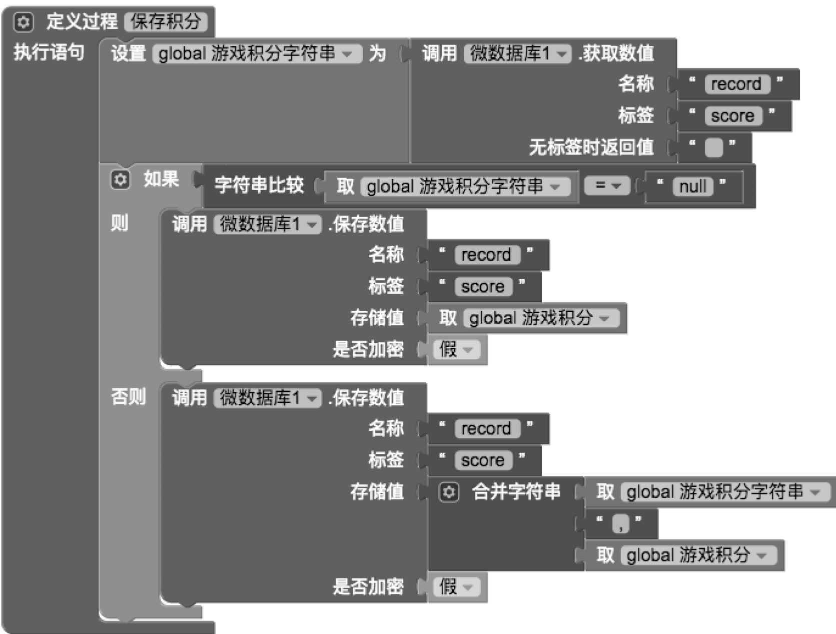


图 20-21 “保存积分”过程的代码

```
record.xml ×
1  <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2  <map>
3      <string name="score">&quot;14,23,2,11,10,13&quot;</string>
4  </map>
```

图 20-22 保存游戏积分文件内容

⑤“青蛙”图像精灵的功能代码如图 20-23 所示。



图 20-23 “青蛙”图像精灵的功能代码

⑥“查看记录”标签的功能代码如图 20-24 所示。

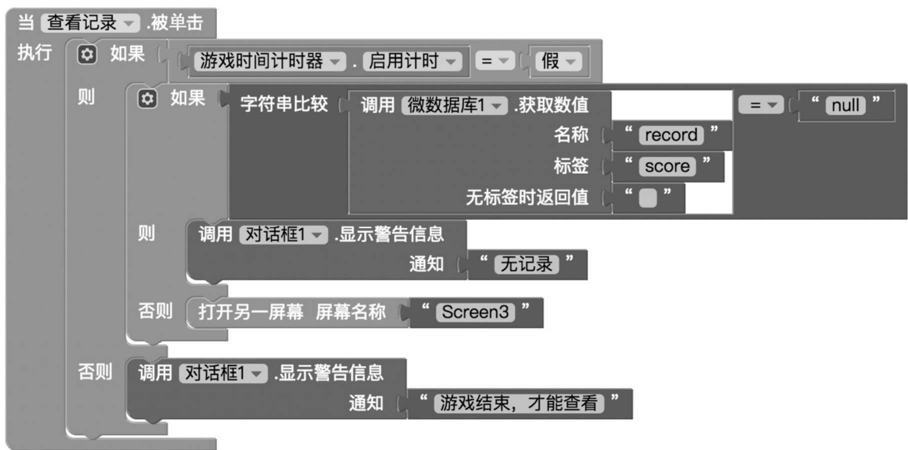


图 20-24 “查看记录”标签的功能代码

⑦“设置”菜单的功能代码如图 20-25 所示。

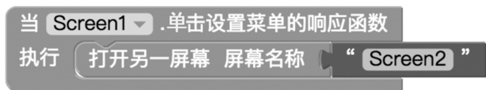


图 20-25 抓青蛙游戏“设置”菜单的功能代码

单击“设置”菜单，会打开“Screen2”屏幕，就是设置背景音的屏幕。

⑧“更新版本”菜单的功能代码如图 20-26 所示。



图 20-26 抓青蛙游戏“更新版本”菜单的功能代码

单击“更新版本”菜单时，设备如处于飞行模式状态，或未连接网络状态，会调用“对话框 1”显示选择对话框，用户单击提示框的“确定”按钮，则会调用“SystemSettings1”组件，打开系统设置界面。代码如图 20-27 所示。



图 20–27 抓青蛙游戏主屏幕选择对话框的功能代码

⑨“Screen1”屏幕状态变更时的处理代码如图 20–28 和图 20–29 所示。

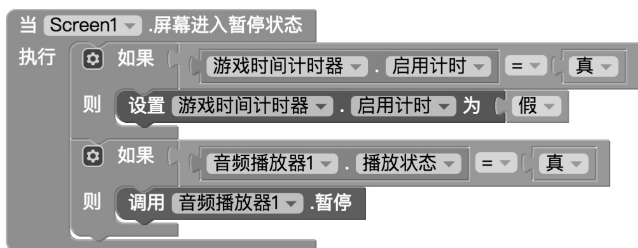


图 20–28 抓青蛙游戏主屏幕进入暂停状态时的功能代码

（当在“Screen1”屏幕打开“Screen2”屏幕或“Screen3”屏幕时，“Screen1”屏幕进入暂停状态；在“Screen1”屏幕按 Home 键，切换 App 到后台时，“Screen1”屏幕也会进入暂停状态。）



图 20–29 抓青蛙游戏主屏幕进入运行状态时的功能代码

（当从“Screen2”屏幕或“Screen3”屏幕返回“Screen1”屏幕时，“Screen1”屏幕进入运行状态；切换 App 从后台切换到前台时，“Screen1”屏幕也会进入运行状态。）

⑩ 电量监测的处理代码如图 20–30 所示。



图 20–30 电量监测的处理代码

（因为此应用比较耗电，在“Screen1”屏幕会调用“BatteryState1”组件监测系统的电量状况。）

3. 设置背景音的屏幕界面及功能实现

1）界面布局如图 20-31 所示。

2）界面使用的组件列表，如图 20-32 所示。



图 20-31　抓青蛙游戏设置屏幕界面布局



图 20-32　抓青蛙游戏设置屏幕的组件列表

组件的属性设置见表 20-3。

表 20-3　抓青蛙游戏设置屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen2	标题	设置
列表显示框 1	背景颜色	透明
	元素字串	开启背景音乐, 关闭背景音乐
	文本颜色	黑色

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

① 屏幕初始化时运行的代码如图 20-33 所示。



图 20-33　抓青蛙游戏设置屏幕初始化时运行的代码

②“列表显示框 1”的功能代码如图 20-34 所示。

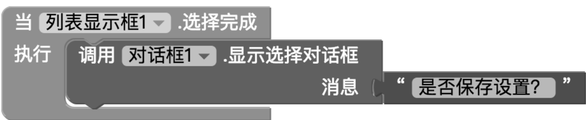


图 20-34　设置屏幕列表显示框的功能代码

③“对话框 1”的功能代码如图 20-35 所示。

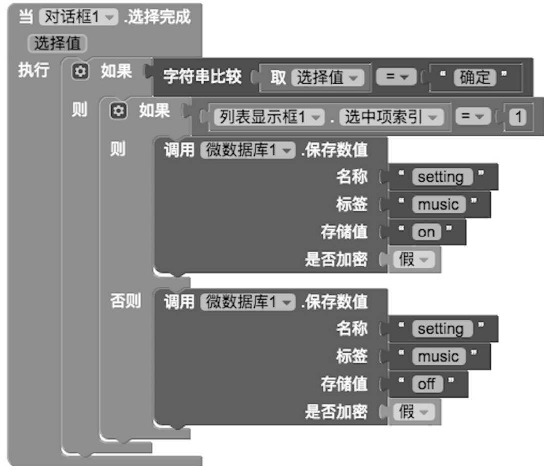


图 20-35　抓青蛙游戏设置屏幕选择对话框的功能代码

4. 查看积分记录的屏幕界面及功能实现

1）界面布局如图 20-36 所示。

2）界面使用的组件列表，如图 20-37 所示。



图 20-36　抓青蛙游戏查看积分屏幕界面布局

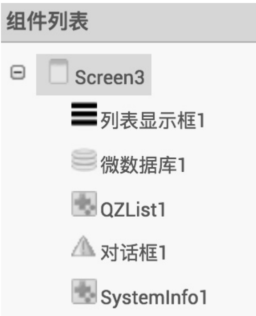


图 20-37　抓青蛙游戏查看积分屏幕的组件列表

组件的属性设置见表 20-4。

表 20-4 抓青蛙游戏查看积分屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen3	标题	积分排行榜
列表显示框 1	背景颜色	透明
	文本颜色	黑色

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

使用到的全局变量和初始值见表 20-5。

表 20-5 抓青蛙游戏查看积分屏幕的全局变量

变量名称	初始值
游戏积分字符串	“”
游戏积分列表	空列表

屏幕初始化时运行的代码如图 20-38 所示。

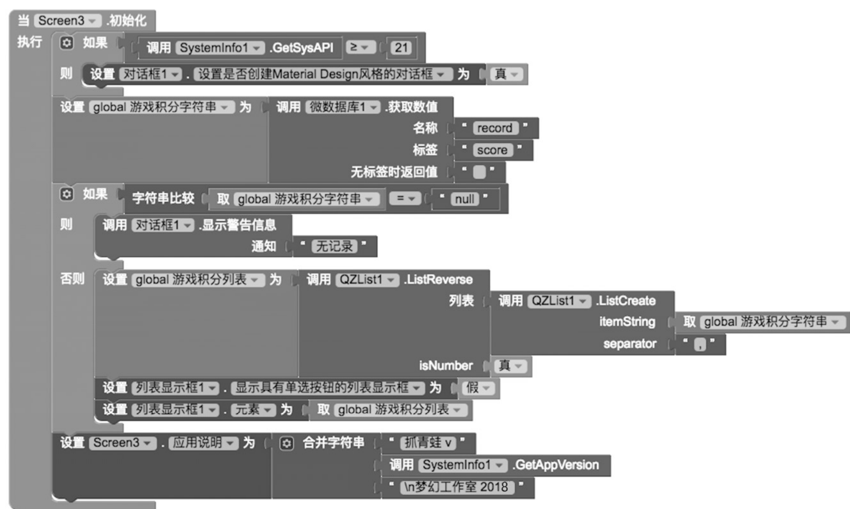


图 20-38 抓青蛙游戏查看积分屏幕的初始化代码

20.2 画图应用

20.2.1 应用简介

此应用提供了以下功能：

1）绘制点、直线、曲线、三角形、矩形、圆角矩形、圆和椭圆等形状，也可绘制文本。

2）可设置画笔的颜色、线条的宽度和文字大小。

3）可以新建和保存图片文件，也可以打开设备上的图片文件进行编辑；还可以把图片文件上传到云端服务器和从云端服务器下载图片。

4）可设置是否填充绘制形状包含的封闭区域和从云端下载图片需要的密码。

20.2.2　应用使用的素材

使用的素材如图 20–39 所示。

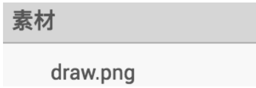


图 20–39　画图应用使用的素材

draw.png——应用在系统桌面显示的 Icon 图片。

20.2.3　应用使用的插件

使用的插件如图 20–40 所示。

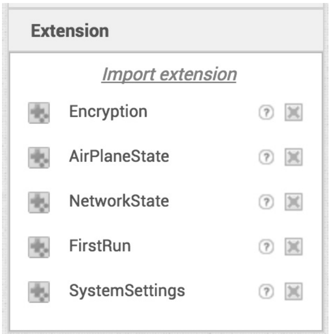


图 20–40　画图应用使用的插件

Encryption——加密功能插件；

AirPlaneState——获取和监测设备飞行模式状态插件；

FirstRun——判断应用是否第一次运行插件；

NetworkState——获取和监测设备网络状态插件；

SystemSettings——系统设置插件。

20.2.4　应用实现

1. 屏幕界面

此应用有两个屏幕：主屏幕（Screen1）和设置屏幕（Screen2）。

图 20–41 为默认的主屏幕界面，图 20–42 是把绘制的内容保存成图片后的主屏幕界面（在屏幕的最上方显示保存文件的全路径）。

主屏幕底部水平显示的组件比较多，如果设备的屏幕比较小，不一定能显示完整，所以使用水平滚动条布局，用户可以左右滑动手指，滚动显示内容。



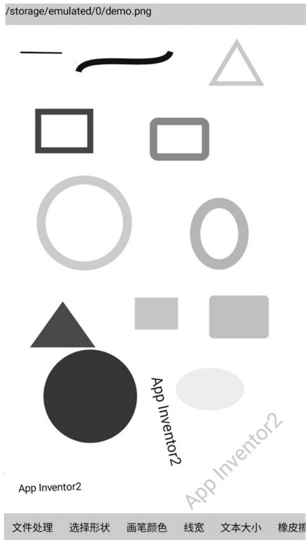


图 20-41　画图应用主屏幕界面 1

图 20-42　画图应用主屏幕界面 2

图 20-42 主屏幕底部最右边组件的文本没有显示完整，可以向左滑动，以便完整显示，如图 20-43 所示。



图 20-43　画图应用主屏幕界面底部

主屏幕有“退出”“关于”“设置”和“更新版本”四个功能菜单，如图 20-44 所示。

图 20-45 是单击“退出”菜单显示的界面，图 20-46 是单击“关于”菜单显示的界面。







图 20-44　画图应用主屏幕功能菜单

图 20-45　单击“退出”菜单显示的界面

单击“设置”菜单，会打开设置屏幕；单击“更新版本”菜单，会打开一个网页。设置屏幕有“退出”和“关于”两个功能菜单，如图 20-47 所示。



图 20-46　单击“关于”菜单显示的界面





图 20-47　画图应用设置屏幕界面

2. 主屏幕界面及功能实现

1）界面布局如图 20-48 所示。

2）界面使用的组件列表，如图 20-49 所示。

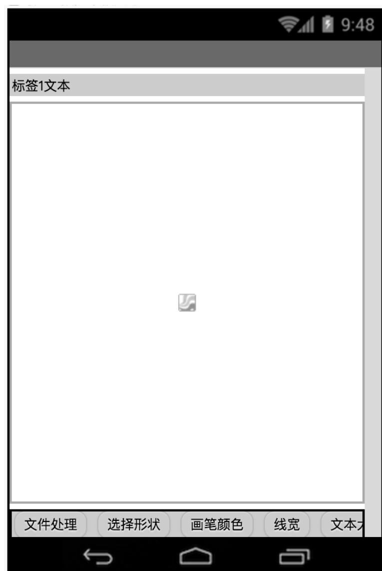


图 20-48　画图应用主屏幕界面布局



图 20-49　画图应用主屏幕的组件列表 1

“水平滚动条布局 1”中包含的组件如图 20-50 所示。



图 20-50　画图应用主屏幕的组件列表 2

组件的属性设置见表 20-6。

表 20-6　画图应用主屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen1	应用说明	画图 v1.0 \n 梦幻工作室 2018
	应用名称	画图
	图标	draw.png
	是否显示设置菜单	是
	是否显示更新版本菜单	是
	状态栏显示	否
	窗口大小	自动调整
	标题展示	否
标签 1	背景颜色	浅灰
	高度	4 比例
	宽度	充满
画布 1	高度	90 比例
	宽度	充满
图像选择框 1	可见性	否
水平滚动条布局 1	垂直对齐	居中
	背景颜色	浅灰
	高度	6 比例
	宽度	充满
文件处理	背景颜色	透明
	元素字串	新建,打开,保存,另存为,上传,下载
	项背景色	白色
	项文本色	黑色
	形状	圆角
	文本	文件处理
	标题	文件处理
选择形状	背景颜色	透明

（续）

组件名称	组件属性	
	属性名称	属性值
选择形状	元素字串	点,直线,曲线,三角形,矩形,圆角矩形,圆,椭圆,文本
	项背景色	白色
	项文本色	黑色
	形状	圆角
	文本	选择形状
	标题	选择形状
画笔颜色	背景颜色	透明
	元素字串	黑色,蓝色,青色,深灰,灰色,浅灰,绿色,品红,橙色,粉色,红色,白色,黄色
	项背景色	白色
	项文本色	黑色
	形状	圆角
	文本	画笔颜色
	标题	画笔颜色
线宽	背景颜色	透明
	形状	圆角
	文本	线宽
文本大小	背景颜色	透明
	形状	圆角
	文本	文本大小
橡皮擦	背景颜色	透明
	形状	圆角
	文本	橡皮擦
网络微数据库 1	服务地址	http://tinywebdb.gzjkw.net/db.php?user=xjbclz&pw=123abc456&v=1

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

① 使用到的全局变量和初始值见表 20-7。

表 20-7　画图应用主屏幕的全局变量

变 量 名 称	初　始　值
文件全路径	未命名
文件名称	未命名
文件处理	“”
文件内容	“”
设置属性	“”
形状	“”
密码	“”

（续）

变 量 名 称	初 始 值
起点 x 坐标	0
起点 y 坐标	0
圆角半径	0
角度	0
线宽	0
画布颜色	0
填充	假
选中橡皮擦	假
画笔颜色列表	（见图 20-51）

“画笔颜色列表”的初始值如图 20-51 所示。

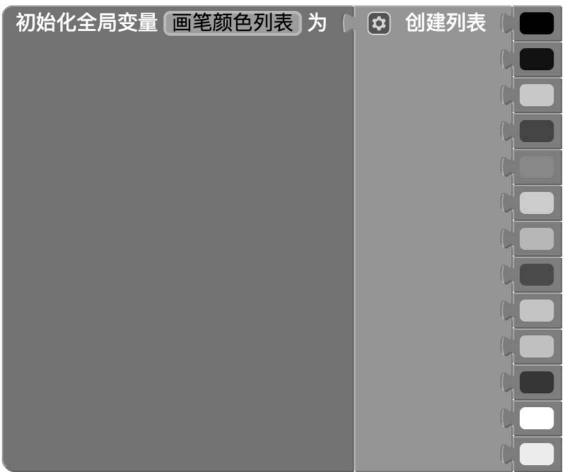


图 20-51 “画布颜色列表”的初始值

② 屏幕初始化时运行的代码如图 20-52 所示。



图 20-52 画图应用主屏幕初始化时运行的代码

在没有保存绘制画布上的内容时，屏幕顶部的标签显示的文件名为“未命名”，提示用户还没有执行过保存操作。

第一次运行应用时，设置不填充绘制形状包含的封闭区域，对应的标签“fill”的值为假。

Android 系统从 API 级别 21 开始才支持 Material Design 风格设计，调用“SystemInfo1”插件，判断当前系统的 API 级别是否大于等于21，如果是则设置“对话框 1”的显示风格为 Material Design 风格。

③ 屏幕进入运行状态时的代码如图 20-53 所示。



图 20-53　画图应用主屏幕进入运行状态时的代码

（屏幕初始化完成后，就进入运行状态；从设置屏幕返回到主屏幕时，也会进入运行状态。）

④ 单击“文件处理”列表显示框，显示如图 20-54 所示界面。



图 20-54　文件处理功能列表界面

对应的功能代码如图 20-55 所示。

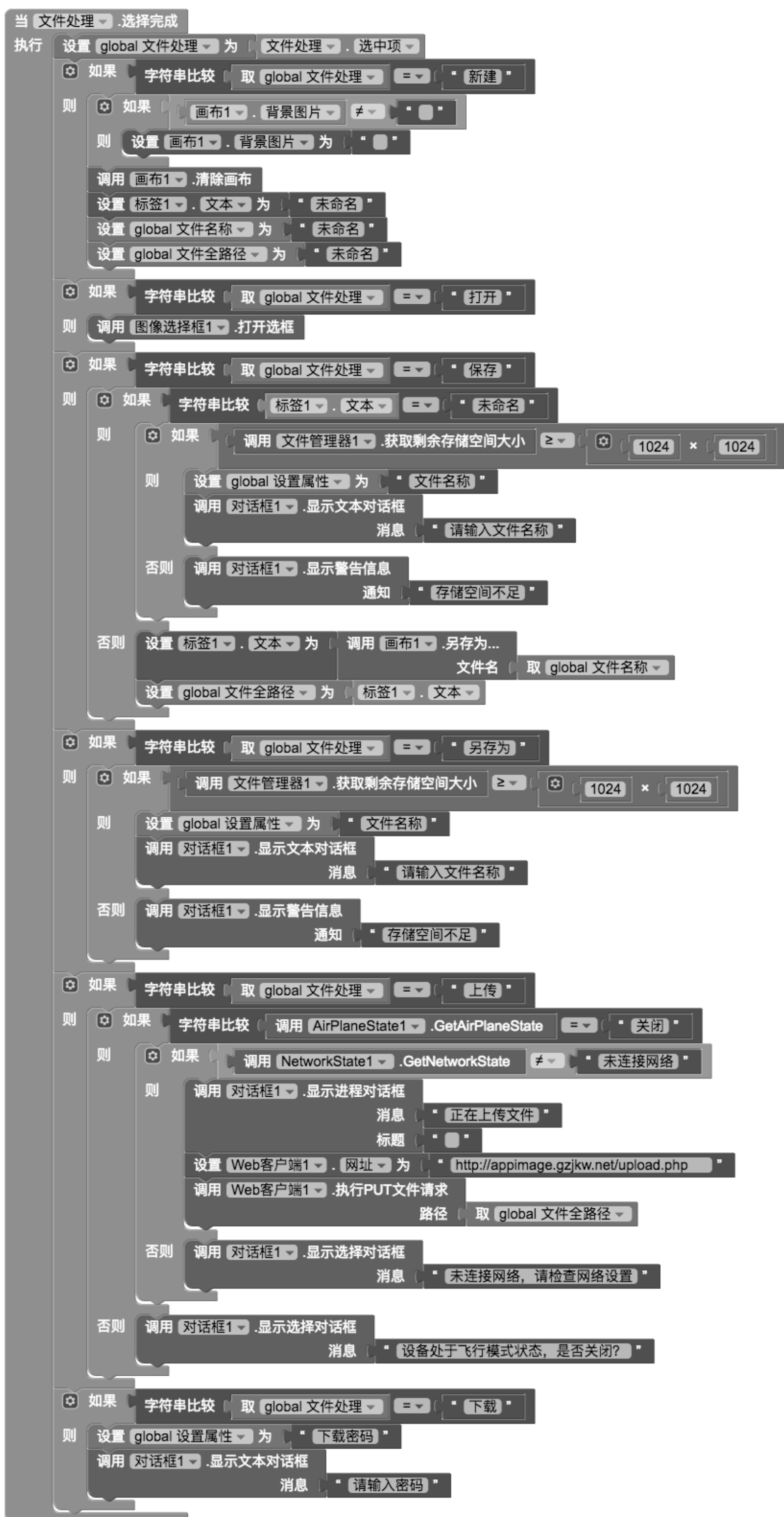


图 20-55 “文件处理”列表显示框的功能代码

选中“打开”选项时，调用“图像选择框 1”，可以选择设备上的图片，选择完成后，把选中的图片设置为“画布 1”的背景图片，代码如图 20-56 所示。



图 20-56　图像选择框的功能代码

图 20-57 为选择图片的界面，图 20-58 为选择完图片后的主屏幕界面。

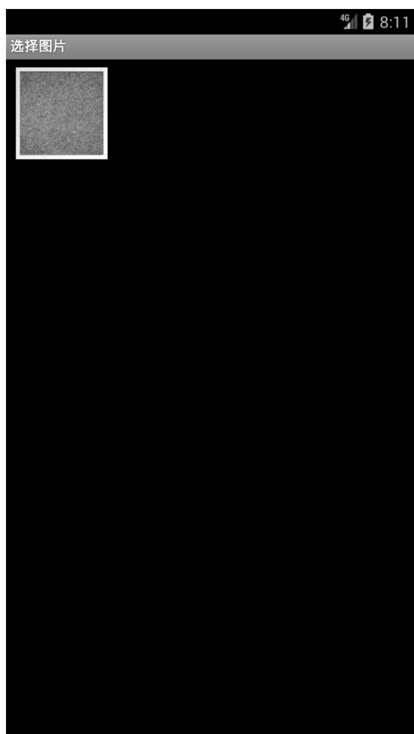


图 20-57　选择图片的界面

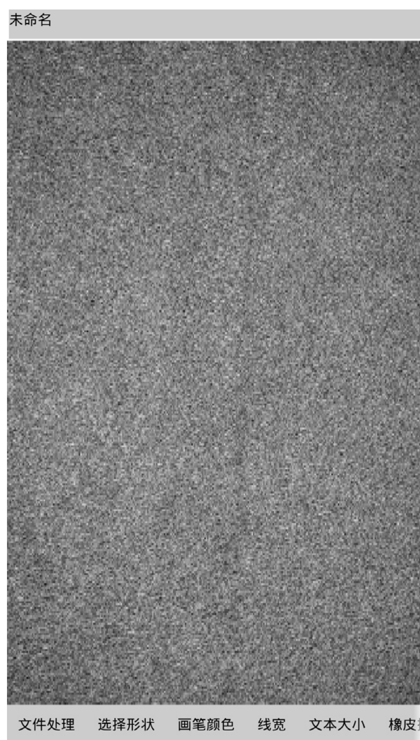


图 20-58　选择完图片后的主屏幕界面

选中“保存”选项时，如果“标签 1”的文本为“未命名”，说明画布上的内容还未保存成文件，需要有存储空间保存新文件，在保存前获取剩余存储空间大小，只有不小于 1MB 的时候，才能保存文件，且需要用户输入文件名称；如果不为“未命名”，则说明已经保存成文件，不需要用户再输入文件名称。

选中“另存为”选项时，也需要有存储空间保存新文件，在保存前获取剩余存储空间大小，只有不小于 1MB 的时候，才能保存文件，且需要用户输入文件名称。

在输入文件名称时，只需输入主文件名，不需输入文件后缀，应用默认把图片保存成 png 格式。

选中“上传”选项时，把图片上传到云端服务器，网址为 http://appimage.gzjkw.net/upload.php。上传完图片后，把图片的网址保存在网络数据库中，以便后续下载用。代码如

图 20-59 所示。

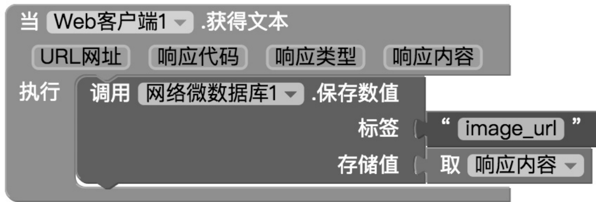


图 20-59　保存图片网址的代码

（把图片和图片网址保存在云端服务器，这样如果用户丢失了设备，或换了设备，就可以从云端服务器获取到图片，而不会丢失图片。）

选中“下载”选项时，需要先输入正确的密码，才能从到云端服务器下载图片。下载完后，自动把图片设置为画布的背景图片。

⑤“网络微数据库 1”的功能代码如图 20-60～图 20-62 所示。

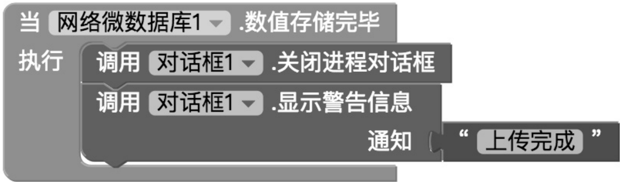


图 20-60　主屏幕网络微数据库的功能代码 1

注：把图片的网址保存到云端服务器后，调用的代码。

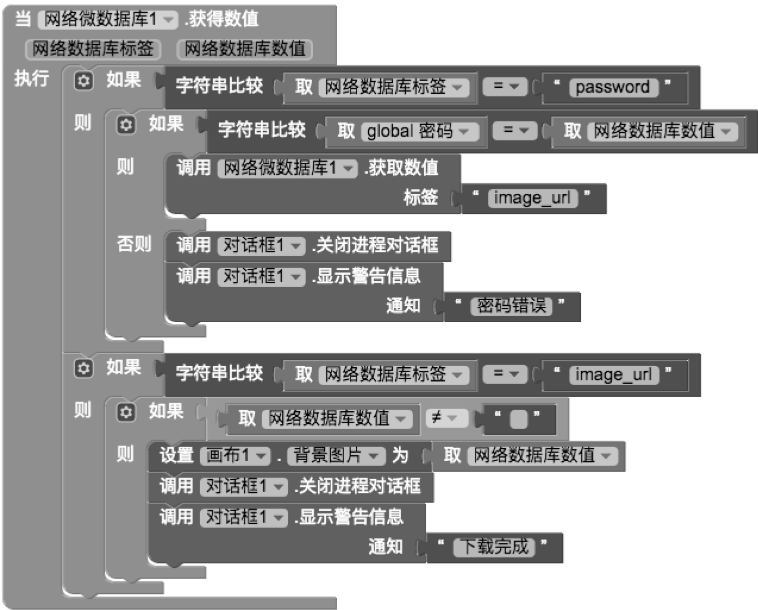


图 20-61　主屏幕网络微数据库的功能代码 2

注：从云端服务器获取密码和图片网址的代码。

⑥ 单击“选择形状”列表显示框，显示如图 20-63 所示界面。

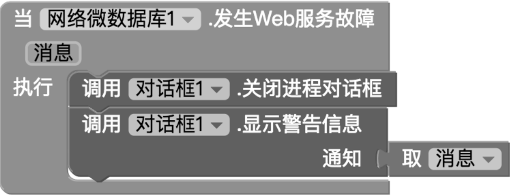


图 20-62　主屏幕网络微数据库的功能代码 3



图 20-63　形状列表界面

对应的功能代码如图 20-64 所示。

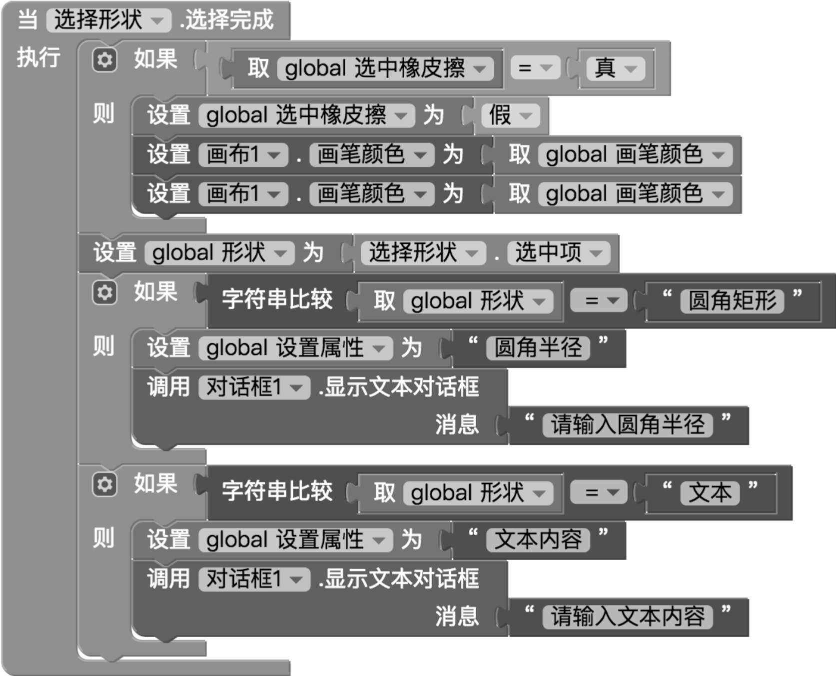


图 20-64　“选择形状”列表显示框的功能代码

⑦ 单击“画笔颜色”列表显示框，显示如图 20-65 所示界面。



图 20-65　颜色列表界面

对应的功能代码如图 20-66 所示。



图 20-66 “画笔颜色”列表显示框的功能代码

⑧“线宽”按钮的功能代码如图 20-67 所示。



图 20-67 “线宽”按钮的功能代码

⑨“文本大小”按钮的功能代码如图 20-68 所示。

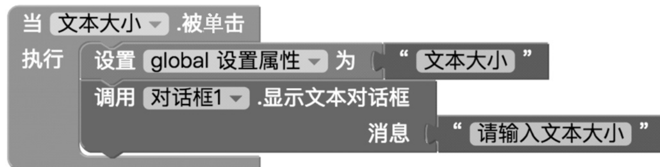


图 20-68 “文本大小”按钮的功能代码

⑩“橡皮擦”按钮的功能代码如图 20-69 所示。

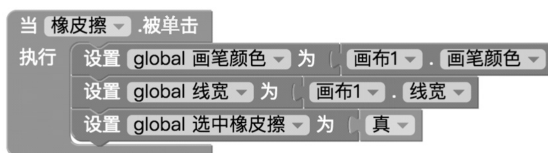


图 20-69 “橡皮擦”按钮的功能代码

使用橡皮擦功能时，会设置画笔颜色为白色，线宽为 8，需要先保存当前的画笔颜色和线宽值，以备后续绘制图形使用。

⑪“对话框 1”文本对话框的功能代码如图 20-70 所示。



图 20-70 画图应用主屏幕文本对话框的功能代码

在使用文本对话框的时候，用户可能忘记输入，直接单击了“确定”按钮，这样会导致应用出现异常，为了避免这个情况发生，增加判断只有当响应内容不是“取消”字符串和空字符串的时候，才会执行后续代码。

⑫“画布 1”的功能代码如图 20-71～图 20-73 所示。

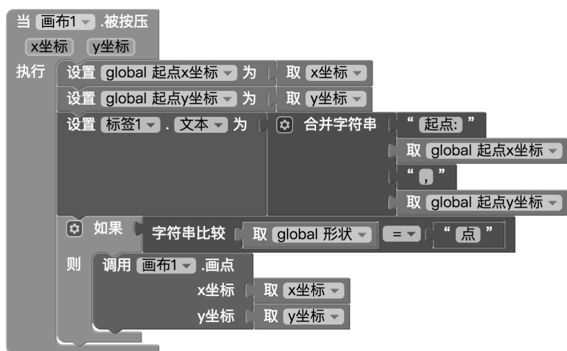


图 20-71 画布被按压时的功能代码

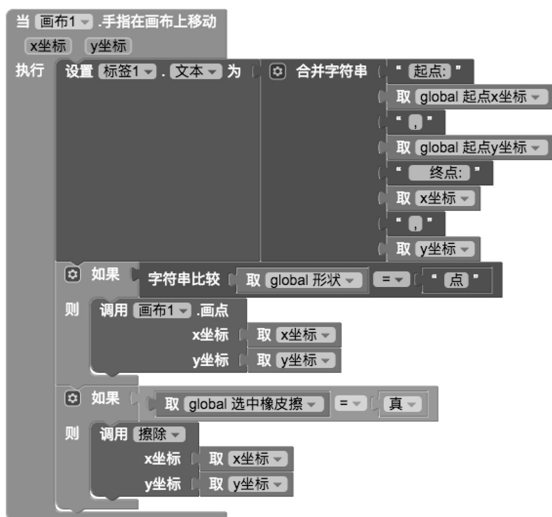


图 20-72 手指在画布上移动时的功能代码

用户用手指单击屏幕时，在顶部标签上显示当前点的 x 和 y 坐标，作为起点坐标，手指在屏幕上移动时，把当前点的坐标作为终点坐标，显示在顶部标签。手指离开屏幕，意味用户不再绘制，此时在顶部标签上显示文件名称。

⑬“擦除”过程的功能代码如图 20-74 所示。

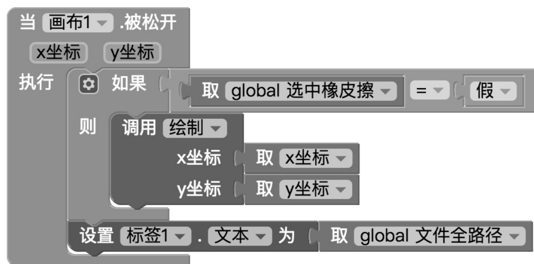


图 20-73 画布被松开时的功能代码



图 20-74 “擦除”过程的功能代码

擦除功能就是利用在屏幕上绘制白色的直线实现的。

⑭“绘制”过程的功能代码如图 20-75 所示。

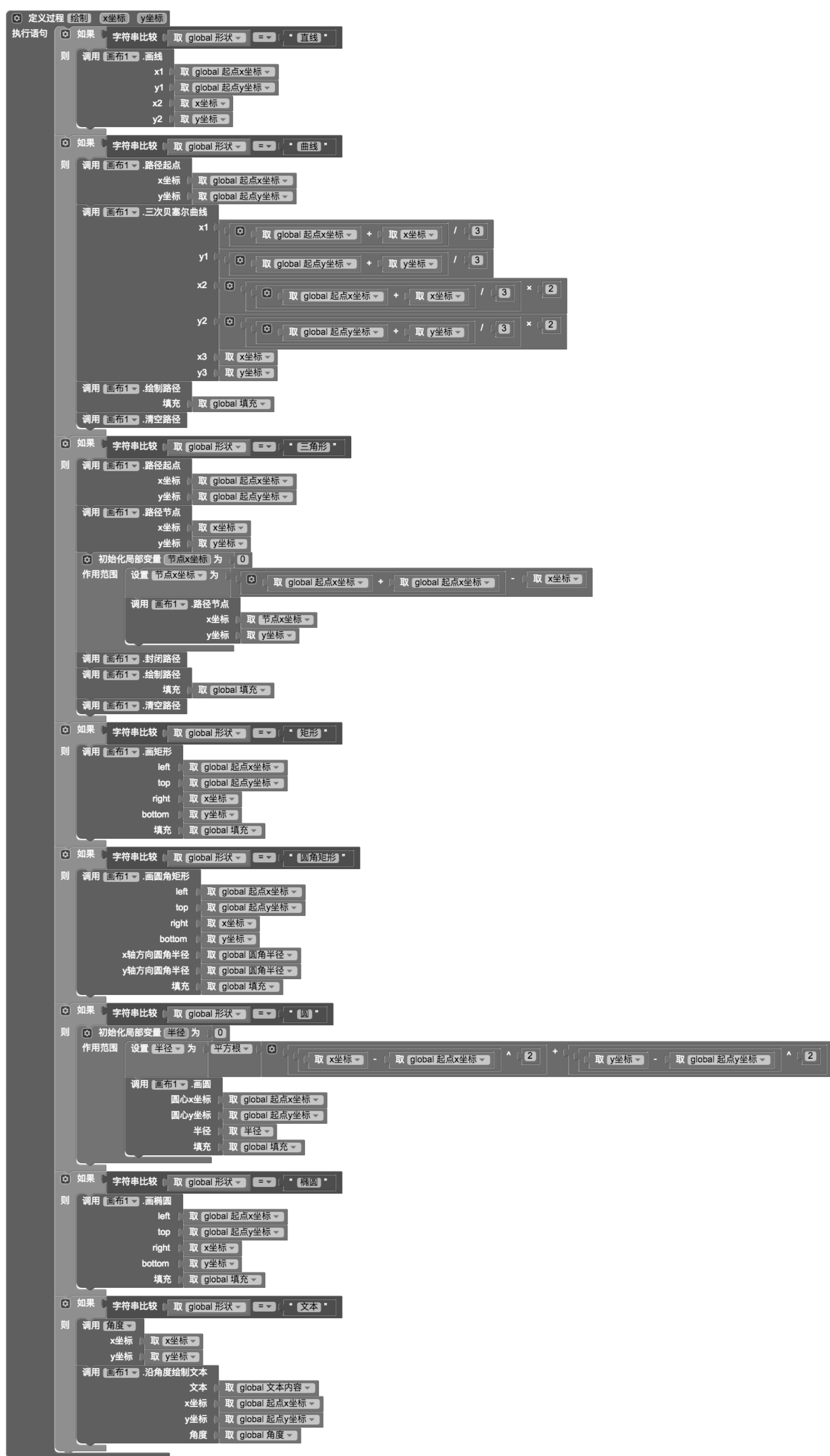


图 20-75 “绘制”过程的功能代码

⑮“角度”过程的功能代码如图 20-76 所示。

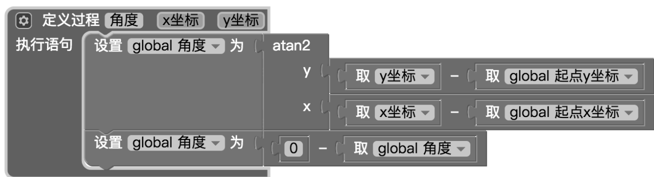


图 20-76 “角度”过程的功能代码

角度值是在绘制文本时用的，因为计算角度和绘制文本时的 y 轴坐标一个向上，一个向下，所以角度值的正负值刚好相反。

⑯“设置”菜单的功能代码如图 20-77 所示。

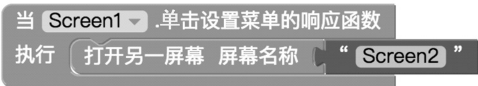


图 20-77 画图应用“设置”菜单的功能代码

单击“设置”菜单，会打开“Screen2”屏幕，就是设置背景音的屏幕。

⑰“更新版本”菜单的功能代码如图 20-78 所示。



图 20-78 画图应用“更新版本”菜单的功能代码

⑱“对话框 1”选择对话框的功能代码如图 20-79 所示。

上传或下载文件，以及单击“更新版本”菜单时，如设备处于飞行模式状态，或未连接网络状态，会调用“对话框 1”显示选择对话框，用户单击提示框的“确定”按钮，则会调用“SystemSettings1”组件，打开系统设置界面。



图 20-79 画图应用主屏幕选择对话框的功能代码

⑲ 异常处理的功能代码如图 20-80 所示。

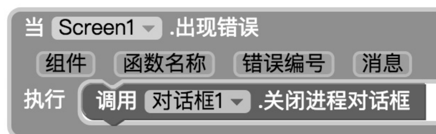


图 20-80 画图应用主屏幕异常处理的功能代码

在应用运行的时候，尤其在上传和下载文件的过程中，涉及网络数据传输，可能会遇到各种异常，此时需要关闭进程对话框，否则，即使应用不能正常运行了，进程对话框还会一直显示。

“出现错误”函数自身会调用对话框组件函数显示出错信息，开发人员不需要再调用函数显示出错信息。函数源码在 Screen 的源码文件 Form.java 中，代码如下：

```
@SimpleEvent(
    description = "Event raised when an error occurs. Only some errors will " +
    "raise this condition.   For those errors, the system will show a notification " +
    "by default.   You can use this event handler to prescribe an error " +
    "behavior different than the default.")
public void ErrorOccurred(Component component, String functionName, int errorNumber,
    String message) {
  String componentType = component.getClass().getName();
  componentType = componentType.substring(componentType.lastIndexOf(".") + 1);
  Log.e(LOG_TAG, "Form " + formName + " ErrorOccurred, errorNumber = " + errorNumber +
      ", componentType = " + componentType + ", functionName = " + functionName +
      ", messages = " + message);
  if ((!(EventDispatcher.dispatchEvent(
      this, "ErrorOccurred", component, functionName, errorNumber, message)))
      && screenInitialized)   {
    // If dispatchEvent returned false, then no user-supplied error handler was run.
    // If in addition, the screen initializer was run, then we assume that the
    // user did not provide an error handler.    In this case, we run a default
    // error handler, namely, showing a notification to the end user of the app.
    // The app writer can override this by providing an error handler.
    new Notifier(this).ShowAlert("Error " + errorNumber + ": " + message);
  }
}
```

3. 设置屏幕界面及功能实现

1）界面布局如图 20-81 所示。

2）界面使用的组件列表，如图 20-82 所示。



图 20-81　画图应用设置屏幕界面布局



图 20-82　画图应用设置屏幕的组件列表

组件的属性设置见表 20-8。

表 20-8　画图应用设置屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen1	应用说明	画图 v1.0 \n 梦幻工作室 2018
	标题	设置
标签 1	背景颜色	浅灰
	宽度	充满
	文本	填充选项
列表显示框 1	背景	白色
	元素字串	填充区域,不填充区域
	文本颜色	黑色
	字号	16
标签 2	背景颜色	浅灰
	宽度	充满
	文本	设置密码
水平布局 1	垂直对齐	居中
	宽度	充满
密码输入框 1	宽度	60 比例
	提示	请输入 8 位数字或字母密码
	文本长度	8
复选框 1	宽度	40 比例

（续）

组件名称	组件属性	
	属性名称	属性值
复选框 1	文本	显示密码
水平布局 2	宽度	充满
清空	宽度	50 比例
	文本	清空
保存	宽度	50 比例
	文本	保存
网络微数据库 1	服务地址	http://tinywebdb.gzjkw.net/db.php?user=xjbclz&pw=123abc456&v=1

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

① 使用到的全局变量和初始值见表 20-9。

表 20-9　画图应用设置屏幕的全局变量

变量名称	初始值
密码显示	假
旧密码	“”

②“列表显示框 1”的功能代码如图 20-83 所示。



图 20-83　画图应用设置屏幕列表显示框的功能代码

③“复选框 1”的功能代码如图 20-84 所示。



图 20-84　复选框的功能代码

④“清空”按钮的功能代码如图 20-85 所示。

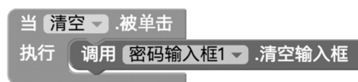


图 20-85 “清空”按钮的功能代码

⑤“保存”按钮的功能代码如图 20-86 所示。

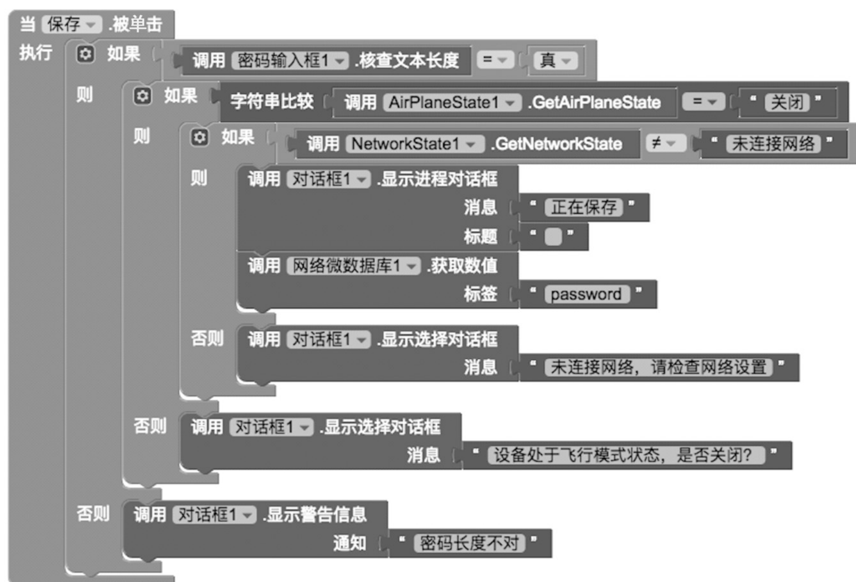


图 20-86 “保存”按钮的功能代码

用户单击“保存”按钮时，首先判断密码长度是否满足要求，只有长度满足，才执行后续 代码。

⑥“网络微数据库 1”的功能代码如图 20-87～图 20-89 所示。



图 20-87 设置屏幕网络微数据库的功能代码 1

在保存密码时，首先从网络微数据库中获取密码，如果获得的密码长度不是32（在保存密码时，把密码用 MD5 加密，转化成长度是32的字符串），则认为是首次设置密码，直接保存密码；否则显示文本对话框，让用户输入旧密码，只有旧密码输入正确，才能保存新密码。

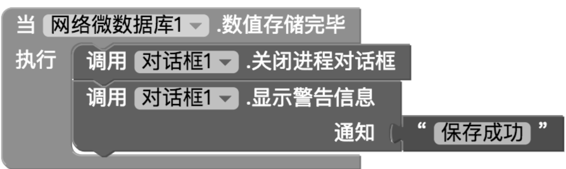


图 20-88 设置屏幕网络微数据库的功能代码 2

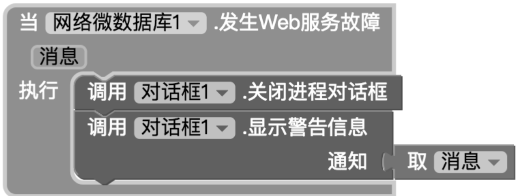


图 20-89 设置屏幕网络微数据库的功能代码 3

⑦“对话框 1”的功能代码如图 20-90 和图 20-91 所示。



图 20-90 画图应用设置屏幕对话框的功能代码 1

在使用文本对话框的时候，用户可能忘记输入，直接单击了“确定”按钮，因为密码长度为 8，这样密码肯定不对，就不需要和旧密码进行比较了；所以增加判断，只有当响应内容不是“取消”字符串和空字符串的时候，才会执行后续代码。

⑧ 异常处理的功能代码如图 20-92 所示。

在应用运行的时候，尤其在保存密码的过程中，涉及网络数据传输，可能会遇到各种异常，此时需要关闭进程对话框，否则，即使应用不能正常运行了，进程对话框还会一直显示。



图 20-91　画图应用设置屏幕对话框的功能代码 2

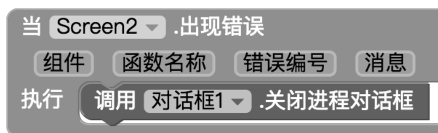


图 20-92　画图应用设置屏幕异常处理的功能代码

20.3　定位应用

20.3.1　应用简介

此应用可设置监测对象所处在的地理围栏，并实时监测对象的位置是否处在地理围栏之内，如果不在地理围栏之内，会自动拨打设置的通知手机号码和向此手机号发送短信报警，可用于监测人员位置、宠物位置和物体的位置等。

此应用使用了高德开放平台提供的 Web 服务 API 坐标转换、地理/逆地理编码、静态地图和地理围栏服务等 HTTP 接口。

此应用的主要运行流程如图 20-93 所示。

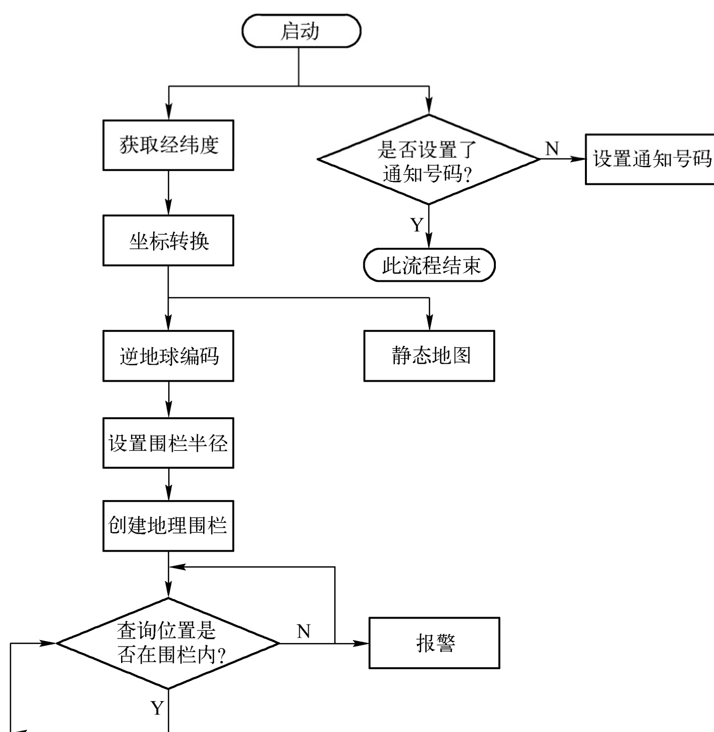


图 20-93　流程图

流程说明：

1）此应用一启动，就调用位置传感器开始定位，同时从微数据库里获取通知手机号码

的数值：如果值不为空，则不做任何处理，此子流程终止；如果值为空，则会显示文本对话框，请用户设置通知手机号码。

2）使用位置传感器获取到的经纬度坐标是 GPS 坐标系坐标，需要调用高德提供的 Web 服务器 API 坐标转换 HTTP 接口，把 GPS 坐标系坐标转换成高德坐标系坐标。

3）获得高德坐标系坐标后，调用高德提供的 Web 服务器 API 逆地理编码 HTTP 接口，通过坐标获取实际的地址信息，并设置成标签文本，展示给用户；以及调用高德提供的 Web 服务器 API 静态地图 HTTP 接口，将高德地图以图片形式嵌入网页中，并在地图上的当前位置处添加标签，显示在 Web 浏览框中。

4）获取到实际的地址信息，并设置成标签文本，展示给用户后，用户就可设置地理围栏半径。设置完成后，调用高德提供的 Web 服务器 API 创建地理围栏 HTTP 接口，创建一个圆形围栏：中心点的坐标是当前位置坐标，半径是用户设置的地理围栏半径。

5）创建完地理围栏后，应用每隔 1min 调用一次高德提供的 Web 服务器 API 围栏设备监控 HTTP 接口，查询当前位置是否在地理围栏内：如果在地理围栏内，则设置标签文本为“当前位置在地理围栏内”，并展示给用户；否则设置标签文本为“当前位置在地理围栏外”，展示给用户，同时自动拨打设置的通知手机号码和向此手机号发送短信报警。

20.3.2 应用使用的素材

使用的素材如图 20-94 所示。



素材
dingwei.png

图 20-94　定位使用使用的素材

dingwei.png——应用在系统桌面显示的 Icon 图片。

20.3.3 应用使用的插件

使用的插件如图 20-95 所示。

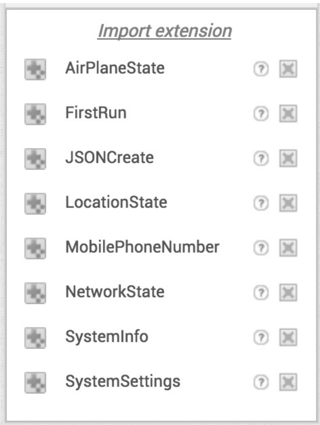


图 20-95　定位使用使用的插件

AirPlaneState——获取和监测设备飞行模式状态插件；
FirstRun——判断应用是否第一次运行插件；
JSONCreate——创建 JSON 字符串插件；
LocationState——获取定位功能状态插件；
MobilePhoneNumber——手机号码校验插件；
NetworkState——获取和监测设备网络状态插件；
SystemInfo——获取设备和系统信息插件；
SystemSettings——系统设置插件。

20.3.4 应用实现

1. 屏幕界面

此应用有一个屏幕：主屏幕（Screen1）。

第一次启动应用时，会现在主屏幕界面显示功能说明，如图 20-96 所示。

功能说明界面显示 5s 后或单击此界面，在主屏幕上显示界面如图 20-97 所示。

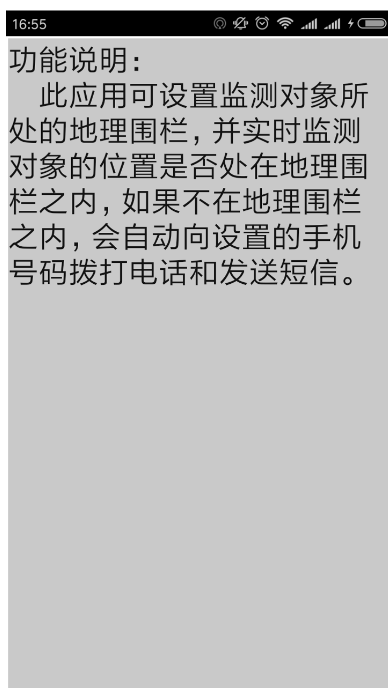


图 20-96　功能说明界面



图 20-97　定位应用主屏幕界面 1

因为必须设置通知手机号码，否则此应用不能起到应有的作用，所以文本输入框只有确定按钮，没有取消按钮。如果设置过通知手机号码，则在启动应用的时候，在主界面不显示此文本输入框。

定位成功后，显示界面如图 20-98 所示。



图 20-98　定位应用主屏幕界面 2

2. 主屏幕界面及功能实现

1）界面布局如图 20-99 所示。

2）界面使用的组件列表如图 20-100 所示。

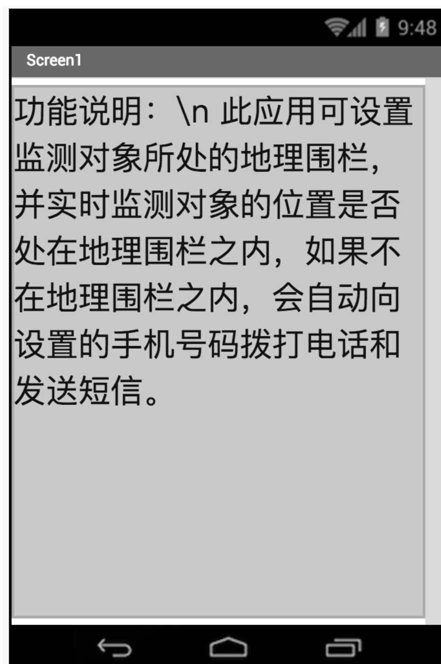


图 20-99　定位应用主屏幕界面布局



图 20-100　定位应用主屏幕的组件列表

组件的属性设置见表 20-10。

表 20-10　定位应用主屏幕的组件属性

组件名称	组件属性	
	属性名称	属性值
Screen1	应用说明	定位 v1.0 \n 梦幻工作室 2018
	应用名称	定位
	图标	dingwei.png
	是否显示设置菜单	是
	是否显示更新版本菜单	是
	窗口大小	自动调整
	标题展示	否
功能说明	背景颜色	青色
	字号	28
	高度	充满

（续）

组件名称	组件属性	
	属性名称	属性值
功能说明	宽度	充满
	文本	功能说明：\n 此应用可设置监测对象所处的地理围栏，并实时监测对象的位置是否处在地理围栏之内，如果不在地理围栏之内，会自动向设置的手机号码拨打电话和发送短信
	文本颜色	蓝色
水平布局 1	宽度	充满
	可见性	否
信息展示	宽度	75 比例
	文本	正在定位中…
功能按钮	启用	否
	文本	重新定位
定位计时器	启用计时	否
	时间间隔	30000
初始计时器	启用计时	否
	时间间隔	5000
轮询计时器	启用计时	否
	时间间隔	60000

注：表中未列出的组件属性，使用系统提供的默认值。

3）功能代码。

① 使用到的全局变量和初始值见表 20-11。

表 20-11　定位应用主屏幕的全局变量

变 量 名 称	初　始　值
经度	0
纬度	0
获取坐标次数	0
地理围栏半径	0
围栏个数	0
gid	“”
设置参数	“”
请求名称	“”
经纬度	“”
服务器返回的数据列表	空列表

② 屏幕初始化时运行的代码如图 20-101 所示。



图 20-101　定位应用初始化代码

在初始化的时候，首先判断是否开启了定位功能，只有开启了定位功能，才会继续运行主流程代码；否则显示选择对话框，提示用户开启定位功能。

③“初始化”过程的功能代码如图 20-102 所示。



图 20-102　“初始化”过程的功能代码

在初始化的时候，首先判断应用是否是第一次运行，如果是第一次运行，则启动“初始计时器”，在屏幕上显示功能说明文字 5s；否则不显示功能说明文字，调用“可见性设置”和“初始化数据”两个过程。

④“初始计时器”的功能代码如图 20-103 所示。

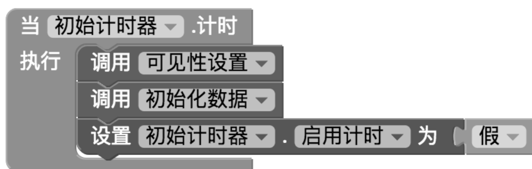


图 20-103　“初始计时器”的功能代码

功能说明文字只显示 5s，所以计时器时间到后，停止计时。

⑤“可见性设置”过程的功能代码如图 20-104 所示。

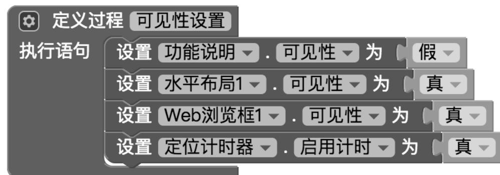


图 20-104　“可见性设置”过程的功能代码

“水平布局 1”和“Web 浏览框 1”两个组件默认不显示，在显示完功能说明文字后，隐藏“功能说明”标签，显示这两个组件，并启动“定位计时器”，轮询定位结果（位置传感

器默认是启用状态）。

⑥“初始化数据”过程的功能代码如图 20-105 所示。

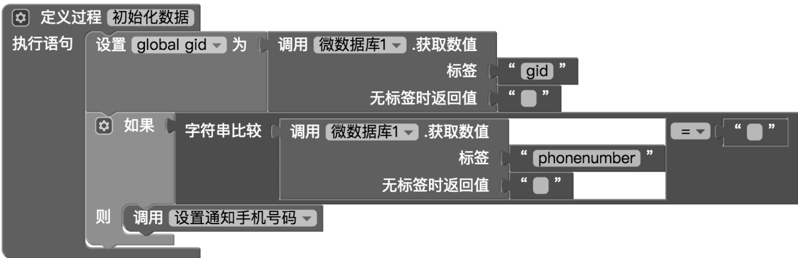


图 20-105 “初始化数据”过程的功能代码

“微数据库 1”的标签“gid”存储的值是创建围栏成功后，服务器返回给 App 的地理围栏的全局 id。

“微数据库 1”的标签“phonenumber”存储的值是通知手机号码，如果值为空，则调用“设置通知手机号码”过程，请用户设置通知手机号码。

⑦“设置通知手机号码”过程的功能代码如图 20-106 所示。



图 20-106 “设置通知手机号码”过程的功能代码

因为必须要设置通知手机号码，否则此应用无法起到应有的作用，所以文本对话框不能被允许撤销。

⑧“查询围栏”过程的功能代码如图 20-107 所示。

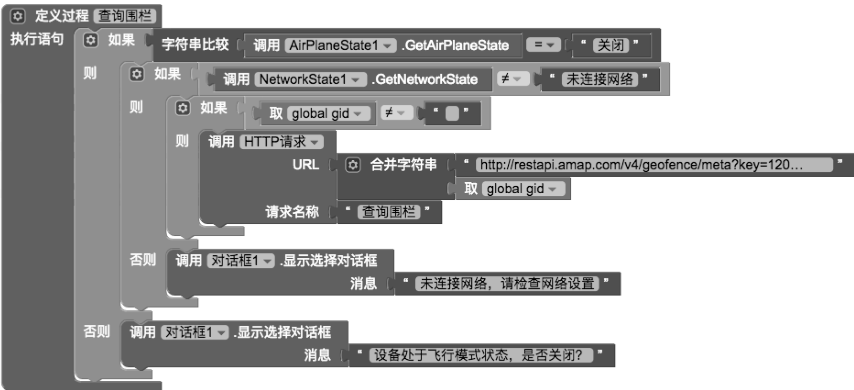


图 20-107 “查询围栏”过程的功能代码

如果用户没有打开定位功能或未连接网络，则会显示选择对话框，提示用户；如果用户单击“确定”按钮，则会跳转到系统设置界面。

如果打开定位功能和连接网络了，且从“微数据库 1”组件中获取的 gid 数据不为空，则会向服务器发送请求，查询此 gid 值对应的地理围栏是否还存在。

图中未显示完整的字符串内容为

http://restapi.amap.com/v4/geofence/meta?key=120eb06f3c657b31293d2a727f959adb&gid=

（其中的 key 值是在高德网站申请的）

⑨“对话框 1”选择对话框的功能代码如图 20-108 所示。



图 20-108　选择对话框的功能代码

⑩“HTTP 请求”过程的功能代码如图 20-109 所示。



图 20-109 “HTTP 请求”过程的功能代码

因为有多处需要向服务器发送 GET 请求，所以封装一个过程函数，以方便使用。参数“请求名称”表明调用的是高德提供的哪个 Web 服务器 API HTTP 接口，在这个过程中，把“请求名称”值赋值给全局变量“请求名称”，以方便在解析服务器返回的数据时使用。

⑪“定位计时器”的功能代码如图 20-110 所示。



图 20-110 “定位计时器”的功能代码

“定位计时器”的时间间隔是 30s，每隔 30s 会查询“位置传感器 1”的纬度和经度数值是否都是 0，如果两个都不是 0，则停止计时，并向服务器发送请求把 GPS 坐标数值转换成高德使用的坐标数值。

累计共轮询 10 次，也就是 5min，如果 5min 后还不能获取到经度和纬度坐标，则停止计时，提示用户“定位失败”，并且“功能按钮”变为可单击状态，用户可单击此按钮重新定位。

图中未显示完整的字符串内容为

http://restapi.amap.com/v3/assistant/coordinate/convert?key=120eb06f3c657b31293d2a727f959adb&locations=

⑫“功能按钮”的功能代码如图 20-111 所示。



图 20-111 “功能按钮”的功能代码

此功能按钮有两个作用：重新定位和设置地理围栏半径，用户单击此按钮时，根据此按钮的文本，执行不同的代码。

⑬“对话框 1”文本对话框的功能代码如图 20-112 所示。

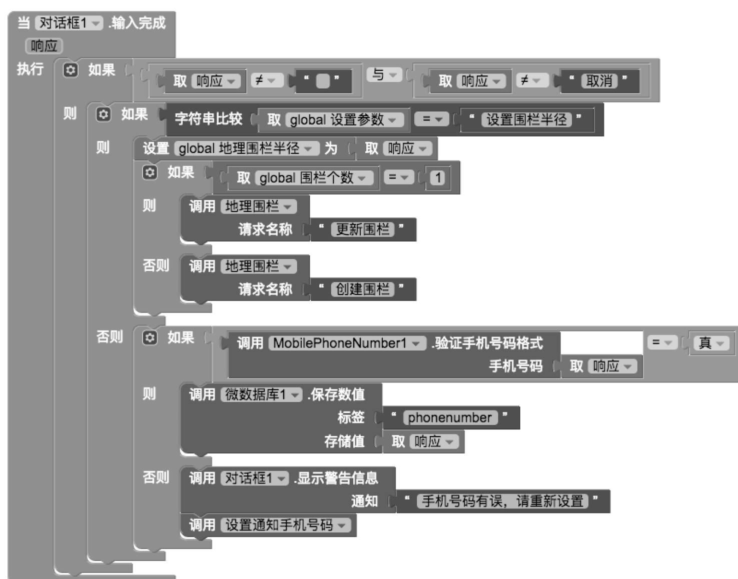


图 20-112 “对话框 1”文本对话框的功能代码

在此应用中，有两个地方用到文本对话框：设置围栏半径和设置通知手机号码，全局变量“设置参数”用于标识是设置哪个参数。“Screen1”屏幕进入运行状态时会调用高德提供的 Web 服务器 API 查询围栏 HTTP 接口，查询是否已经设置过地理围栏，并把服务器返回的围栏个数值赋值给全局变量“围栏个数”。

如果此变量的值为 1，说明之前已经设置过围栏，在设置完围栏半径后，调用高德提供的 Web 服务器 API 更新围栏 HTTP 接口，更新围栏数据；否则，调用创建围栏 HTTP 接口，创建新围栏。

因为必须要设置通知手机号码，否则此应用无法起到应有的作用，所以在用户设置通知手机号码失败后，会再调用“设置通知手机号码”过程，让用户设置通知手机号码。

⑭“地理围栏”过程的功能代码如图 20-113 所示。

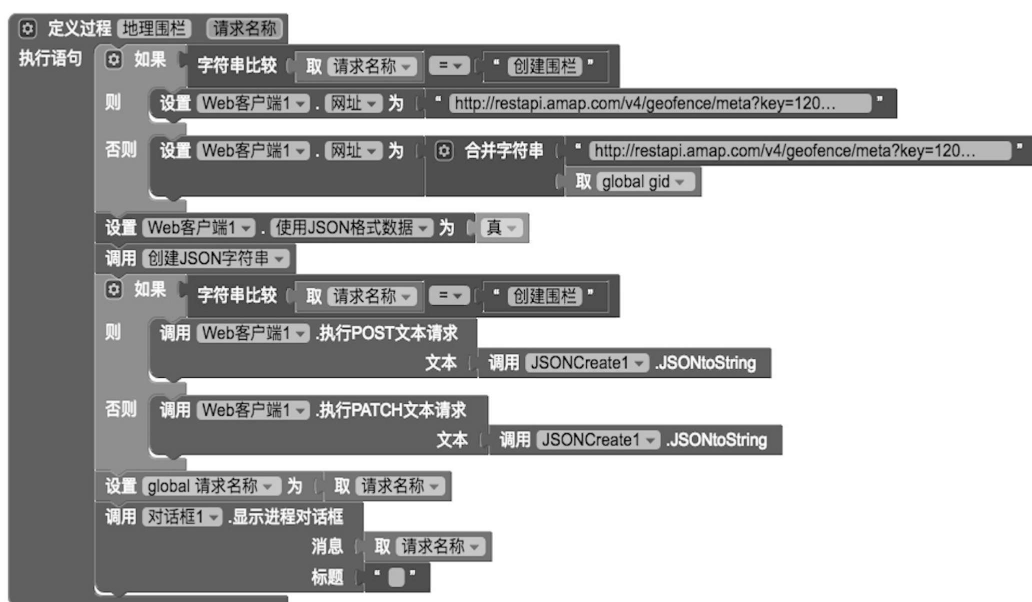


图 20-113 “地理围栏”过程的功能代码

创建围栏是调用高德提供的 Web 服务器 API 查询围栏 HTTP 接口，更新围栏是调用高德提供的 Web 服务器 API 更新围栏 HTTP 接口，两者的代码类似，都需要向服务器发送 JSON 文本，所以把代码合并为一个过程函数。

这个过程的参数“请求名称”用于标识是创建围栏还是更新围栏。

图中创建围栏时，未显示完整的字符串内容为

http://restapi.amap.com/v4/geofence/meta?key=120eb06f3c657b31293d2a727f959adb

更新围栏时，未显示完整的字符串内容为

http://restapi.amap.com/v4/geofence/meta?key=120eb06f3c657b31293d2a727f959adb&gid=

⑮“创建 JSON 字符串”过程的功能代码如图 20-114 所示。

⑯“Web 客户端 1”的功能代码如图 20-115 所示。

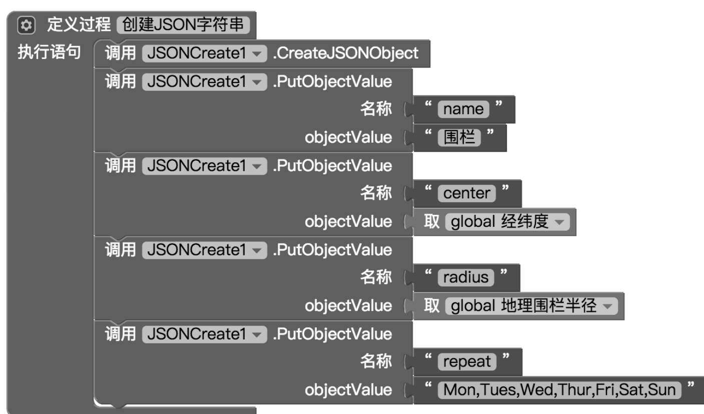


图 20-114 “创建 JSON 字符串”过程的功能代码



图 20-115 “Web 客户端 1”的功能代码

⑰“解析数据”过程的功能代码如图 20-116 所示。



图 20-116 “解析数据”过程的功能代码

在此过程中，全局变量“请求名称”标识服务器是响应高德提供的哪个 Web 服务器 API HTTP 接口返回的数据。

具体的数据解析代码如图 20-117～图 20-122 所示。

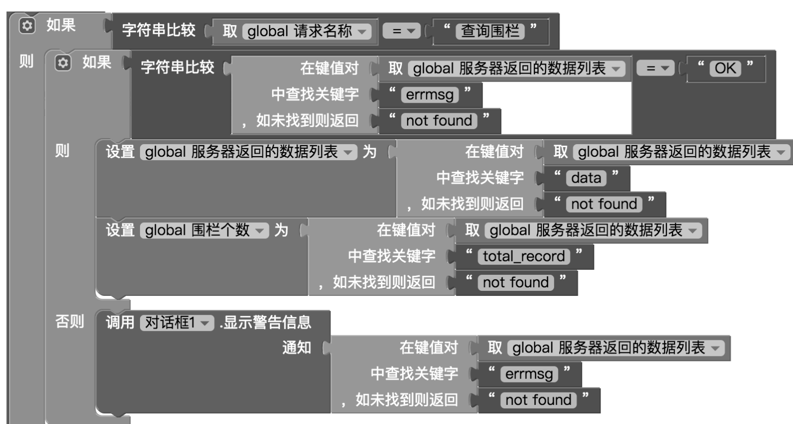


图 20-117　数据解析代码 1

调用高德提供的 Web 服务器 API 查询围栏 HTTP 接口向服务器发送请求后，如请求成功，服务器返回数据中的“total_record”字段数值表明了是否存在围栏，图 20-117 中代码把“total_record”字段数值赋值给了全局变量“围栏个数”，以方便在其他地方使用。

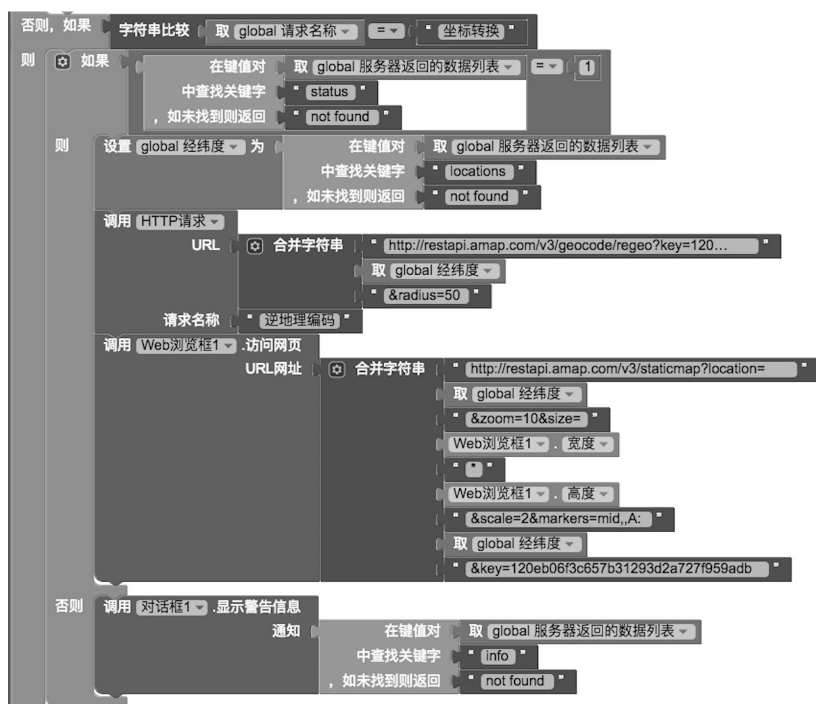


图 20-118　数据解析代码 2

调用高德提供的 Web 服务器 API 坐标转换 HTTP 接口向服务器发送请求后，如请求成功，服务器返回数据中的“locations”字段数值就是转换后的经纬度。

在获取到转换后的经纬度后，调用 Web 服务器 API 逆地理编码 HTTP 接口和静态地图 HTTP 接口，向服务器发起请求。

图中未显示完整的字符串内容为

http://restapi.amap.com/v3/geocode/regeo?key=120eb06f3c657b31293d2a727f959adb&location=

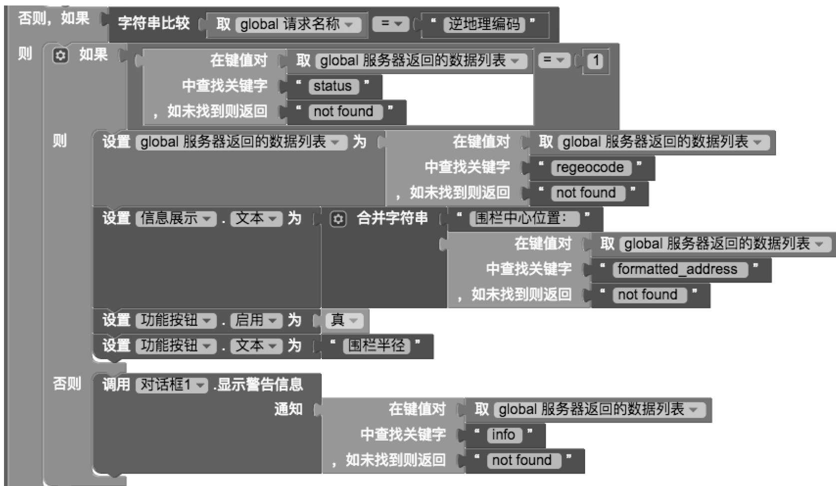


图 20-119　数据解析代码 3

调用高德提供的 Web 服务器 API 逆地理编码 HTTP 接口向服务器发送请求后，如请求成功，服务器返回数据中的“formatted_address”字段数值就是当前位置的具体地址信息。

在获取到地址信息后，设置“功能按钮”的状态为启用，按钮文本设置为“围栏半径”，此时用户可设置地理围栏半径。



图 20-120　数据解析代码 4

调用高德提供的 Web 服务器 API 创建围栏 HTTP 接口向服务器发送请求后，如请求成功，服务器返回数据中的“gid”字段数值就是围栏全局 id。调用“微数据库 1”组件，把全局 id 保存在设备中，以便下次使用应用时，利用此 id 查询是否存在地理围栏。

一旦创建了地理围栏，就启用“轮询计时器”，每隔 1min 查询下当前位置是否在地理围栏内。

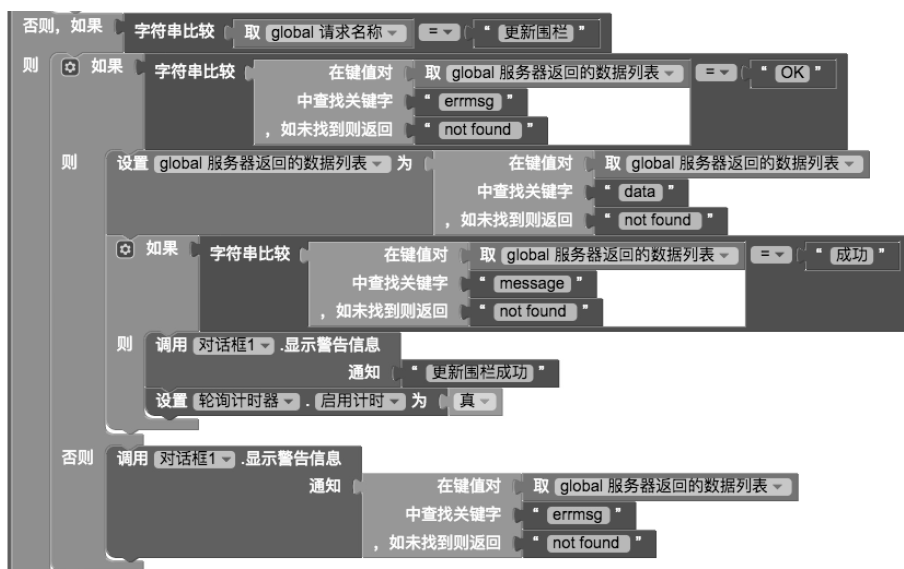


图 20-121 数据解析代码 5

更新围栏成功后，也要启用“轮询计时器”，每隔 1min 查询下当前位置是否在地理围栏内。

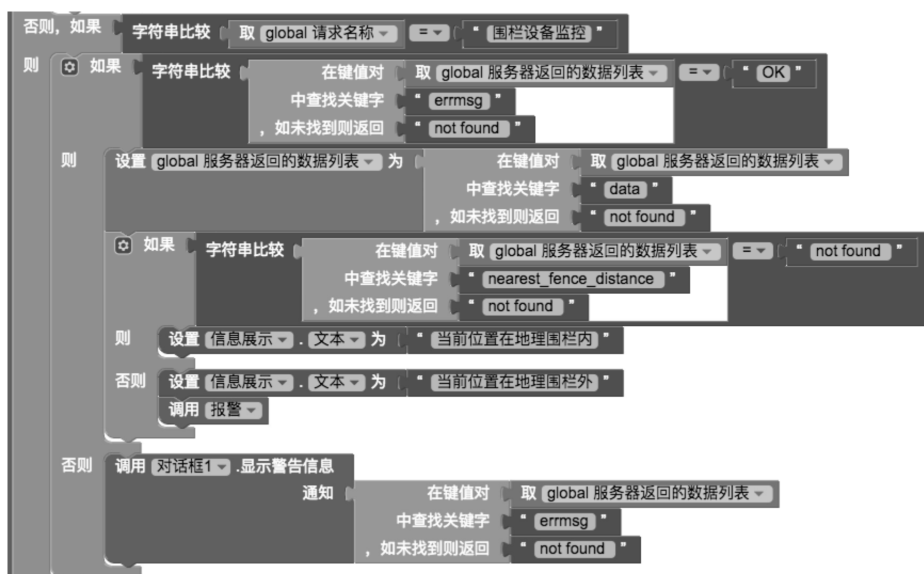


图 20-122 数据解析代码 6

调用高德提供的 Web 服务器 API 创建围栏 HTTP 接口向服务器发送请求后，如请求成功，服务器返回数据中如果没有“nearest_fence_distance”字段，表明当前位置在地理围栏内，否则表明当前位置在地理围栏外，此时调用“报警”过程，向设置的手机号码拨打电话和发送短信。

⑱“轮询计时器”的功能代码如图 20-123 所示。



图 20-123 “轮询计时器”的功能代码

图中未显示完整的字符串内容为

http://restapi.amap.com/v4/geofence/status?key=120eb06f3c657b31293d2a727f959adb&diu=

⑲“报警”过程的功能代码如图 20-124 所示。



图 20-124 “报警”过程的功能代码

⑳“设置”菜单的功能代码如图 20-125 所示。

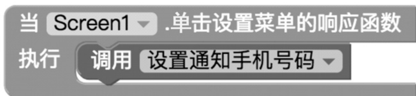


图 20-125 定位应用“设置”菜单的功能代码

㉑“更新版本”菜单的功能代码如图 20-126 所示。

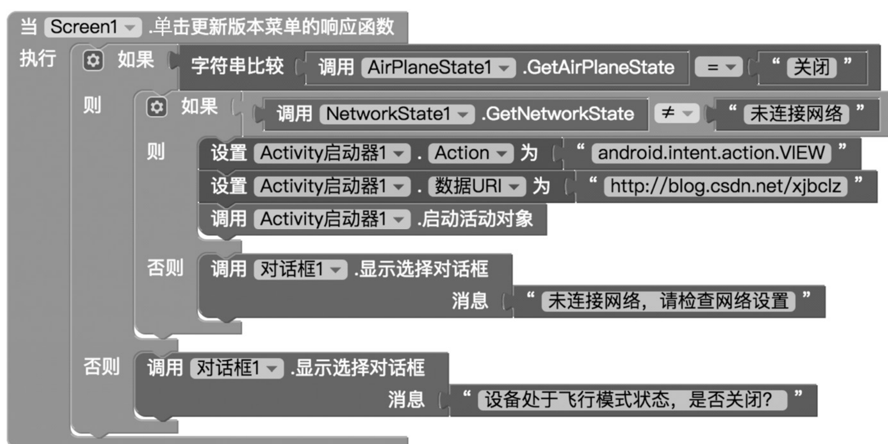


图 20-126　定位应用“更新版本”菜单的功能代码

㉒ 异常处理的功能代码如图 20-127 所示。



图 20-127　定位应用主屏幕异常处理的功能代码

在应用运行的时候，尤其是在和服务器交互的过程中，可能会遇到各种异常，此时需要关闭进程对话框，否则，即使应用不能正常运行，进程对话框还会一直显示。

在线互动交流平台

官方微博：http://weibo.com/cmpjsj
豆 瓣 网：http://site.douban.com/139085/
读者信箱：cmp_itbook@163.com

内容简介

本书介绍了App Inventor 2开发涉及的Android相关知识，App Inventor 2的源码、自带组件的功能扩展和插件的开发，并提供了大量实例，使读者对App Inventor 2开发相关知识有一个全面了解，能够对App Inventor 2进行定制与二次开发，并在此基础上，能够以更专业和高效的方式开发出UI界面更丰富、功能更强大的应用。

本书的目标读者是从事App Inventor 2开发教育、培训工作的教师和培训人员，学习App Inventor 2开发的学生和开发人员等。无论是App Inventor 2的初学者，还是中高级读者，本书都是一本不可多得的开发秘籍。


计算机分社微信服务号

关注计算机分社官方微信，回复您购买图书书号中间的五位数字，即可获取本书配套资源下载链接，并可获得更多增值服务和最新资讯。

ISBN 978-7-111-**59828**-2

地址：北京市百万庄大街22号
邮政编码：100037
电话服务
服务咨询热线：010-88379833
读者购书热线：010-88379469
网络服务
机工官网：www.cmpbook.com
机工官博：weibo.com/cmp1952
金书网：www.golden-book.com
教育服务网：www.cmpedu.com
封面无防伪标均为盗版


机械工业出版社微信服务号

上架指导 计算机/移动开发
ISBN 978-7-111-59828-2
策划编辑◎汤 枫 / 封面设计◎




ISBN 978-7-111-59828-2
9 787111 598282
定价：59.00元